UNIVERSITÀ DEGLI STUDI DI PISA
Facoltà di Ingegneria
Corso di Laurea Specialistica in
Ingegneria Informatica

Curriculum sistemi industriali e sistemi web

Tesi di laurea:

«Progettazione e sviluppo di una piattaforma integrata per l'elaborazione di log di dispositivi remoti»

Relatori:
Prof. Antonio Cosimo Prete
Prof. Pierfrancesco Foglia
Ing. Antonio Raimondo

Candidato:
Marco Chiodetti

ANNO ACCADEMICO 2010/2011

Alla mia famiglia e alla mia ragazza Irene
per aver avuto fiducia in me

Ringraziamenti

Desidero innanzitutto ringraziare il mio relatore Ing. Antonio Raimondo che mi ha permesso di realizzare una tesi nell'ambito del Web, argomento che da sempre mi appassiona. Ringrazio la Dott.ssa Caterina Guazzelli per avermi aiutato durante la stesura della tesi e per i preziosi consigli che mi ha saputo dare.

Ringrazio i miei genitori, per avermi sostenuto in questi anni di studi ed avermi spronato a non mollare mai. Grazie a mia sorella Silvia per tutte le serate trascorse insieme in aula studio e per avermi trasmesso il suo ottimismo. Un grazie di cuore va a mia nonna Anna che si è sempre presa cura di me senza farmi mancare nulla, permettendomi di concentrarmi sugli esami e di raggiungere questo importante traguardo.

Un ringraziamento speciale va alla mia ragazza Irene per aver avuto la pazienza di ascoltarmi quando ripetevo prima di un esame, di sorreggermi nei momenti di difficoltà e di svegliarmi presto tutte le mattine. Grazie per avermi aspettato: finalmente possiamo iniziare la nostra vita insieme.

Un particolare ringraziamento va ai miei compagni di studio, in ordine alfabetico perché tutti allo stesso modo importanti, Alessandro, Pierluca e Riccardo. Insieme abbiamo saputo affrontare mille difficoltà, in mezzo ad esami, ripassi e progetti anche fino all'alba. Spero con tutto il cuore che resteremo sempre una bella squadra!

Riassunto analitico

Abstract

Progettazione e sviluppo di una piattaforma integrata per l'elaborazione di log di dispositivi remoti

Design and development of an integrated platform for the remote device log processing

Lo scopo di questa tesi è stato il progetto e lo sviluppo del software GestDevice e della sua applicazione nell'ambito del progetto UniPOS dell'Università di Pisa. L'idea nasce dall'esigenza di interfacciarsi con dispositivi remoti di natura eterogenea al fine di ricavarne informazioni di stato e diagnostiche oltre ai dati in output per i quali sono utilizzati. GestDevice riceve messaggi in ingresso dai suddetti dispositivi sottoforma di file di testo (log) ed utilizza una lista espandibile di interpreti per comprendere le informazioni in essi contenute. Tali informazioni vengono memorizzate all'interno di una base di dati in modo da essere disponibili per fini statistici. Un'altra caratteristica peculiare di GestDevice è l'efficiente gestione a matrice degli utenti con assegnamento dei privilegi completamente personalizzabile, oltre alla capacità di impostare attività programmate con lo scopo principale di schedulare verifiche sullo stato di una determinata classe di dispositivi. Nello specifico caso d'uso di UniPOS, l'introduzione di GestDevice permette di monitorare lo stato dei POS attivi nell'Ateneo.

The purpose of this thesis has been the design and the development of GestDevice software and its introduction within the UniPOS project of University of Pisa. The basic idea comes from the necessity of an interface with mixed remote devices, in order to gather status and diagnostic information as well as the produced output data. GestDevice receives input messages as text files (logs) from such devices and applies an extensible list of interpreters in order to understand the contained information. Such information are stored into a database to be available for later computing. The set of functions offered by GestDevice includes an efficient matrix-

based user management with a fully customizable privileges allocation and it includes the capability to plan activities with the main purpose of scheduling status tests of a whole class of devices. In the specific UniPOS use case, GestDevice has been introduced in order to monitor the status of POS devices assigned to the faculties.

Sommario

1. Introduzione 13

1.1 Le caratteristiche 14

1.2 Vincoli tecnologici 16

1.3 Caso d'uso: la realtà UniPOS 17

2. Analisi e specifica dei requisiti 18

2.1 Analisi e specifica dei requisiti utente 18

2.1.1 Identificazione degli utenti 18

2.1.2 Funzionalità offerte dal sistema 19

2.2 Analisi e specifica dei requisiti di sistema 23

2.2.1 Core di sistema 25

2.2.2 Modulo di gestione degli utenti 25

2.2.3 Modulo di gestione dei privilegi 28

2.2.4 Modulo di gestione dei dispositivi 30

2.2.5 Modulo di gestione delle attività programmate 34

2.2.6 Modulo di log delle attività 36

2.2.7 Tecnologie utilizzate in fase di implementazione 37

2.2.8 Usabilità dell'interfaccia 43

2.2.9 Sicurezza e consistenza del sistema 45

2.2.10 Tolleranza ai guasti 47

3. Progettazione del sistema 49

3.1 Tipologie di attori 49

3.1.1 Utente generico 50

3.1.2 Operatore 51

3.1.1 Sviluppatore 54

3.1.2 Amministratore ... 56
3.1.3 Super Amministratore ... 60
3.2 Struttura dell'applicazione e mappe di navigazione ... 62
3.2.1 Struttura statica ... 62
3.2.2 Layout grafico ... 66
3.2.3 Interazione tramite AJAX e JSON ... 68
3.2.4 Mappe di navigazione ... 70
3.3 Progetto software ... 77
3.3.1 Classi per la rappresentazione di oggetti ... 77
3.3.2 Progettazione del database ... 84
3.3.3 MDB2 ... 89
3.3.4 Core ... 91
3.3.5 Sicurezza ... 95
4. Il caso d'uso UniPOS ... 96
4.1 Requisiti ... 98
4.1.1 Catalogazione Terminali ... 99
4.1.2 Caratteristiche funzionali ... 100
4.1.3 Configurazione POS ... 101
4.1.4 Analisi statistica attività ... 101
4.1.5 Verifiche e anomalie di sistema ... 101
4.1.6 Guasti Hardware ... 103
4.1.7 Account Utenti ... 103
4.2 Personalizzazione del sistema ... 104
4.2.1 Tipologie di utenti ... 104
4.2.2 Messaggi POS e interpreti ... 105

4.2.3 Attività programmate personalizzate 108

4.2.4 Assegnamento multiplo dei privilegi 109

4.2.5 Localizzazione dispositivi 110

5. Conclusioni 111

Appendice A Codice sorgente delle classi per la rappresentazione di oggetti ... 113

A.1 class.db.php 113

A.2 class.content.php 115

A.3 class.content_attribute.php 120

A.4 class.content_attribute_link.php 124

A.5 class.content_attribute_value.php 127

A.6 class.content_type.php 132

A.7 class.content_relation.php 137

A.8 class.content_enum.php 139

Appendice B Progettazione della base di dati con MySQL Workbench 142

B.1 Instanziare un server 143

B.2 Creare una connessione verso il DB 143

B.3 Disegnare uno schema E-R 143

Appendice C La libreria MDB2 144

Appendice D Manuale dello sviluppatore per UniPOS 146

D.1 Core di sistema 146

D.1.1 File di configurazione 146

D.1.2 Variabili persistenti 147

D.2 Utenti 147

D.2.1 Creazione di un utente 148

D.2.2 Assegnamento dei privilegi 150

D.2.3 Limitazione di accesso alle funzioni personalizzate 151
D.3 Dispositivi 151
D.3.1 Creazione di un tipo di dispositivo 151
D.3.2 Creazione di un attributo 152
D.3.3 Creazione di un dispositivo 156
D.3.4 Gestione della struttura organizzativa 156
D.4 Attività programmate 157
D.4.1 Creazione di un'attività programmata 157
D.5 Personalizzazione 159
D.5.1 Gestione dei comandi del menu 159
D.5.2 Creazione di un gruppo di comandi 160
D.5.3 Creazione di un comando 161
D.5.4 Creazione di un comando non di menu 164
D.5.5 Creazione di un privilegio personalizzato 165
D.6 File di log ed interpreti 166
D.6.1 Impostare la cartella dei file di log 166
D.6.2 Creazione di un interprete 166
Sitografia 168

Indice delle figure

Figura 1 – Ricezione dei messaggi, notifica agli utenti ed attuazione di un'operazione 14
Figura 2 - Struttura a livelli di GestDevice 15
Figura 3 - Rappresentazione UML degli utenti del sistema 19
Figura 4 – Composizione per moduli della piattaforma 24
Figura 5 – Esempio di personalizzazione della piattaforma per un utilizzo specifico 25
Figura 6 - Esempio di contesto organizzativo di una azienda 30
Figura 7 - Esempio di definizione di una struttura organizzativa 32
Figura 8 - Esempio di localizzazione dei dispositivi 33
Figura 9 - Schema di funzionamento del sistema 34
Figura 10 - Attori del sistema 49
Figura 11 - Caso d'uso: sessione utente 50
Figura 12 - Caso d'uso: profilo personale 51
Figura 13 - Caso d'uso: gestione dei dispositivi 52
Figura 14 - Caso d'uso: operazioni con privilegio 53
Figura 15 - Caso d'uso: attività programmate 54
Figura 16 - Caso d'uso: personalizzazione comandi 55
Figura 17 - Caso d'uso: scrittura di un interprete 56
Figura 18 - Caso d'uso: amministrazione utenti 57
Figura 19 - Caso d'uso: amministrazione gruppi 59
Figura 20 - Caso d'uso: amministrazione privilegi 60
Figura 21 - Caso d'uso: gestione unità organizzative 61
Figura 22 - Class diagram di una pagina generica 63
Figura 23 - Mock-up strutturale dell'interfaccia 64
Figura 24 - Composizione dell'intestazione 65
Figura 25 - Composizione del menu di navigazione 65
Figura 26 - Ruota cromatica 67
Figura 27 - Accostamento di colori proposto 67
Figura 28 - Elementi grafici jQuery UI 68

Figura 29 - Mappa di navigazione: accesso 71
Figura 30 - Mappa di navigazione: profilo personale 72
Figura 31 - Mappe di navigazione: gestione utenti 73
Figura 32 - Mappa di navigazione: gestione dispositivi 74
Figura 33 - Mappa di navigazione: attività programmate 75
Figura 34 - Mappa di navigazione: personalizzazione comandi 76
Figura 35 - Mappa di navigazione: privilegi 77
Figura 36 - Diagramma UML delle classi per la rappresentazioine di oggetti generici 79
Figura 37 - Diagramma delle classi per i gestori di attributo 84
Figura 38 - Diagramma E-R della base di dati 87
Figura 39 - Schema di utilizzo delle variabili persistenti 94
Figura 40 - Dispositivi POS inviano messaggi in differenti formati 97
Figura 41 - Attributo "contesto" nel profilo utente 105
Figura 42 - Interfaccia di assegnamento multiplo dei privilegi 109
Figura 43 - Funzionalità di localizzazione dispositivi 110
Figura 44 - Schermata di avvio di MySQL Workbench 142

Indice delle tabelle

Tabella 1 - Esempio di tipi di dispositivo 32

Tabella 2 - Statistiche di utilizzo dei vari browser 44

Tabella 3 - Statistiche di utilizzo di JavaScript 70

Tabella 4 - Vincoli di integrità referenziale 88

1. Introduzione

In un'era tecnologica nella quale l'ottimizzazione del lavoro e la gestione efficace delle risorse sono sempre più importanti, ci sono ancora molte aziende che non riescono a tenere sotto controllo in modo semplice questi aspetti. Il tempo impiegato dal personale dei servizi informatici delle grandi aziende per effettuare la manutenzione dei dispositivi aziendali risulta eccessivo. I controlli, da eseguire sistematicamente ed in modo ciclico, interrompono l'attività lavorativa dei loro utilizzatori e, solo in un numero limitato di casi, si rivelano realmente necessari.

Oggigiorno la quasi totalità dei dispositivi utilizzati nelle aziende ha la capacità di comunicare in rete sia tramite rete cellulare sia tramite connessione cablata che senza fili. La possibilità di sfruttare questa connettività per effettuare il salvataggio su server dei file di log di tali dispositivi ci porta all'idea di progettare e realizzare un sistema automatico di raccolta informazioni e di richiesta di intervento verso i tecnici addetti alla manutenzione. Inviando informazioni diagnostiche ad un sistema centralizzato è possibile interpretare le informazioni contenutevi all'interno per capire se il dispositivo in questione è funzionante o richiede un intervento da parte dei servizi informatici.

L'oggetto di questa tesi tesi è la realizzazione di un sistema informatico in grado di gestire le comunicazioni provenienti da qualsiasi dispositivo utilizzato in ambito aziendale (attrezzature industriali, smartphone, tablet, mezzi di trasporto, ecc.). Tali informazioni sono elaborate da una stazione centrale secondo specifiche politiche definite in fase applicativa.

Data l'ampia gamma di dispositivi ed informazioni da essi ricavate, è necessario dotare il sistema di un'interfaccia per la personalizzazione delle funzioni disponibili per l'utente finale.

Il nome che è stato scelto per questo applicativo è GestDevice e, nel seguito, sarà così riferito.

1.1 Le caratteristiche

All'interno di un contesto aziendale si trovano molteplici tipologie di dispositivi:

- Macchinari addetti alla produzione: rulli trasportatori, robot per catene di produzione, macchine saldatrici, etc;
- Dispositivi per la gestione e la comunicazione: computer desktop, notebook, palmari, tablet, smartphone, etc;
- Mezzi ausiliari: auto aziendali, muletti, tornelli per il controllo degli accessi, etc.

Ognuno di questi dispositivi è in grado di comunicare attraverso messaggi dalla natura fortemente eterogenea sia come formato che come contenuto.

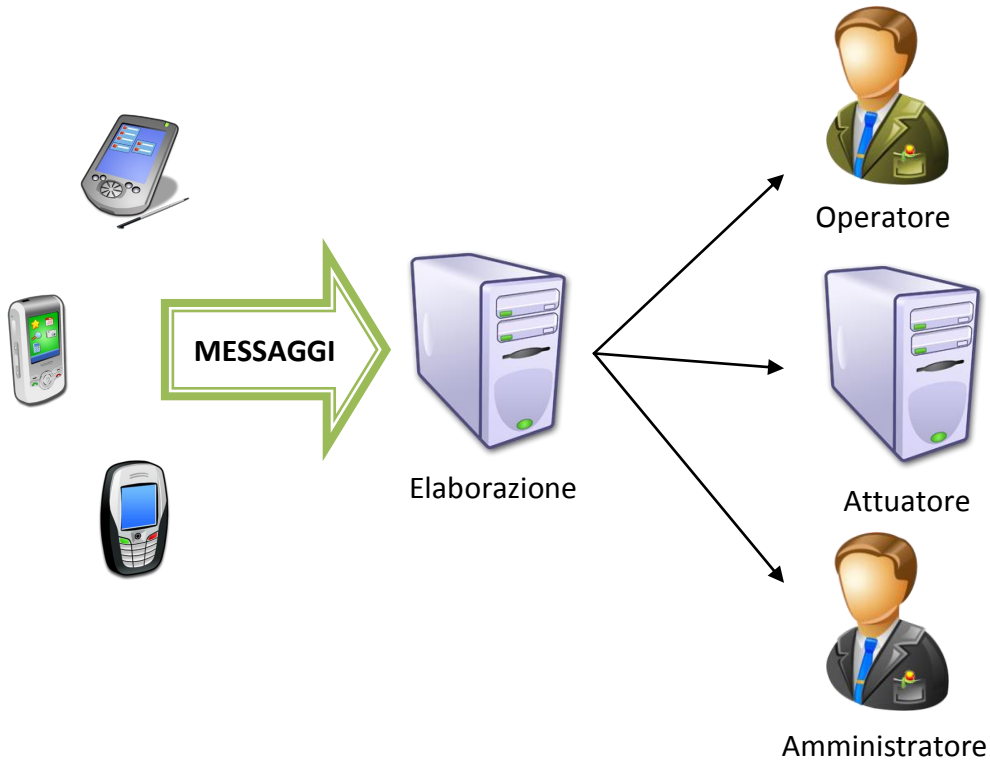


Figura 1 – Ricezione dei messaggi, notifica agli utenti ed attuazione di un'operazione

GestDevice è in grado di comprendere il significato di qualsiasi tipo di messaggio e di estrapolarne le informazioni necessarie ad eseguire successive elaborazioni per fini statistici. Ciò avviene mediante l'impiego di una lista di interpreti, i quali

vengono utilizzati in sequenza fintanto che non viene trovato quello capace di interpretare correttamente il messaggio in questione.

Quindi le informazioni ricavate vengono convogliate verso un server per l'archiviazione e qualsivoglia successivo utilizzo.

Sulla base delle esigenze specifiche dell'utente è possibile adattare il comportamento di GestDevice. Impostando attività programmate si possono definire operazioni da eseguire con la cadenza specificata al fine di:

- mostrare statistiche sui dispositivi;
- creare dei report in PDF;
- diagnosticare situazioni di errore;
- etc.

GestDevice è progettato su due livelli, come si nota in Figura 2: il livello Core è quello che comprende le funzionalità di base del sistema, ossia fornisce gli strumenti per rappresentare qualsiasi tipologia di oggetto (vedi capitolo 3.3.1), gli strumenti per gestire gli utenti, i dispositivi, le attività e i privilegi.

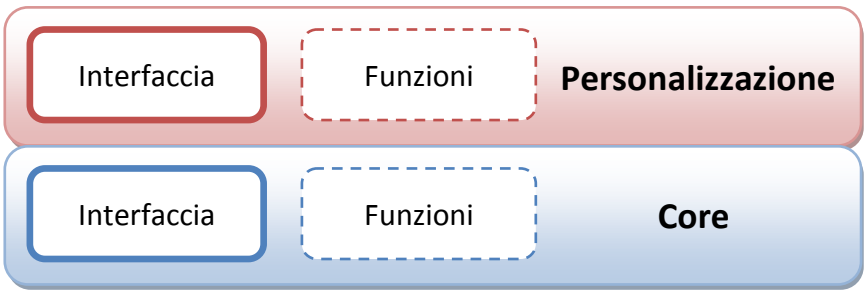


Figura 2 - Struttura a livelli di GestDevice

Il secondo livello è dedicato alla personalizzazione del comportamento di GestDevice al fine di adattarsi a qualsiasi necessità: è possibile definire attività programmate specifiche e creare moduli aggiuntivi per implementare nuove funzionalità.

L'idea alla base di questa suddivisione in due livelli è che il sistema sia aggiornabile da remoto, senza andare però ad intaccare le funzionalità di base che, in ogni installazione, devono essere presenti.

1.2 Vincoli tecnologici

Durante la progettazione è stato necessario porsi il problema dell'ambiente operativo nel quale GestDevice dovrà essere utilizzato: come è noto dall'esperienza, la maggior parte dei sistemi informativi aziendali si basa su sistemi operativi Windows o Linux, ognuno disponibile in varie distribuzioni e versioni.

L'interfaccia e le funzionalità di GestDevice sono state realizzate come Web Application, quindi basate su server Web: i prodotti disponibili sul mercato e in rete sono, fra gli altri, Internet Information Services (IIS) e Apache: il primo è sviluppato da Microsoft ed è disponibile solo su sistemi operativi Windows, mentre il secondo è un software open source, liberamente utilizzabile anche per fini commerciali. Questi due web server presentano caratteristiche similari, ma Apache è disponibile sia per sistemi operativi Windows che Linux: questa caratteristica lo ha reso il candidato per l'ambiente di sviluppo di GestDevice. Tuttavia, in ambiente di produzione, possiamo utilizzare alternativamente Internet Information Services (su macchine Windows) e Apache (su macchine Windows o Linux).

Dal punto di vista della memorizzazione dei dati la scelta si ramifica ulteriormente, in quanto, a livello aziendale, si utilizzano correntemente diversi DBMS, i più diffusi dei quali sono:

- MySQL;
- Microsoft Access;
- Microsoft SQL Server
- IBM DB;
- Oracle.

Ognuno di questi utilizza una versione di SQL molto simile alle altre, ma introduce sempre un certo livello di personalizzazione della sintassi, rendendo impossibile la

scrittura di query sintatticamente comprensibili da qualsiasi DBMS. La libreria denominata MDB2 risolve questo problema in modo trasparente per lo sviluppatore.

1.3 Caso d'uso: la realtà UniPOS

La prima applicazione di GestDevice avviene nel contesto di UniPOS, realtà universitaria dell'Ateneo pisano che nel 2004 ha introdotto la verbalizzazione elettronica degli esami in luogo dei moduli cartacei.

Questa rivoluzione è avvenuta per mezzo dell'introduzione dei dispositivi POS che riducono notevolmente i disagi dovuti alla presenza di errori sugli statini cartacei e ai ritardi di aggiornamento del gestionale universitario.

Purtroppo,però, una rivoluzione comporta sempre qualche disagio: i dispositivi POS, infatti, sono utilizzati solo per la verbalizzazione degli esami e, poiché gli appelli d'esame sono concentrati solo in alcuni periodi dell'anno, i terminali sono lasciati in disuso per settimane o addirittura mesi. Nel momento in cui si ha la necessità di utilizzare un POS, si possono verificare alcuni fastidiosi problemi come: batteria scarica, carta esaurita, guasti hardware, etc.

Per monitorare queste condizioni, il gruppo di sviluppatori che lavora a questo ambizioso progetto utilizza molteplici soluzioni ad hoc, con uno scarso grado di integrazione. Si è manifestata quindi la necessità di disporre di uno strumento integrato che consentisse un maggior controllo sullo stato dei vari POS distribuiti in tutto l'Ateneo pisano.

Per consentire una corretta gestione dei POS, il gruppo UniPOS necessita di una piattaforma per verificare costantemente ed in tempo reale il loro stato. GestDevice viene introdotto in questo ambito proprio a tale scopo.

2. Analisi e specifica dei requisiti

2.1 Analisi e specifica dei requisiti utente

Prima di iniziare un qualsiasi progetto è opportuno disporre di un documento, chiamato documento dei requisiti utente, o URD, in cui sono indicate tutte le funzionalità che il prodotto finito dovrà rendere disponibili.

Il sistema in oggetto è una piattaforma altamente personalizzabile e adattabile a qualsiasi contesto organizzativo, sia pubblico che privato, che consente di monitorare lo stato di dispositivi di qualunque natura (smartphone, cellulari, POS, etc.), attraverso una comoda interfaccia. Tale sistema dovrà essere sviluppato considerando:

- le funzionalità essenziali alla gestione e identificazione dei dispositivi;
- le politiche di accesso alle varie funzioni del sistema;
- la necessità che i dati inviati dai dispositivi siano memorizzati e resi disponibili per elaborazioni successive o analisi personalizzate;
- la possibilità di personalizzare il sistema senza dover in alcun modo mettere mano al core del sistema;
- la necessità che il sistema si mantenga sempre aggiornato.

2.1.1 Identificazione degli utenti

I soggetti che utilizzano il sistema sono molteplici, ognuno con differenti competenze tecniche e informatiche. Sono stati individuati gli utenti, con le relative sigle con cui questi saranno riferiti:

- Super Amministratore (SA): può accedere a tutte le funzionalità offerte dal sistema;
- Amministratore (AM): designato dal Super Amministratore o da un altro Amministratore, può accedere a tutte le funzionalità offerte dal sistema;
- Operatore (OP): usufruisce di limitate funzionalità offerte dal sistema, in base ai privilegi che possiede;

- Sviluppatore (SV): realizza lo strato di software necessario a personalizzare il comportamento della piattaforma.

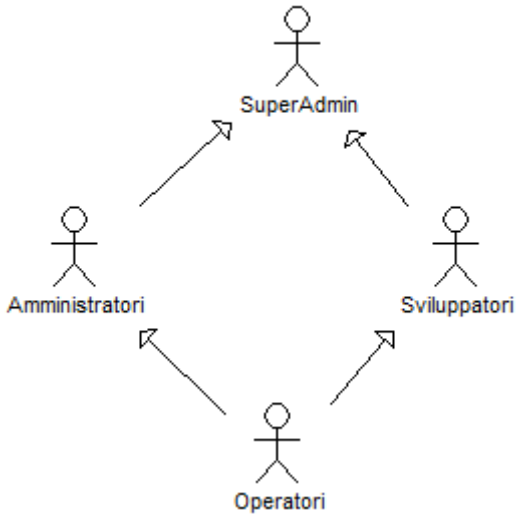


Figura 3 - Rappresentazione UML degli utenti del sistema

Dalla Figura 3 si evincono i rapporti tra gli utenti: un Operatore è un soggetto con limitate funzionalità, un Amministratore ed uno Sviluppatore sono soggetti fidati e non necessitano di privilegi per operare sul sistema. Il Super Amministratore è unico e può svolgere tutte le operazioni messe a disposizione dal sistema.

2.1.2 Funzionalità offerte dal sistema

Numerose sono le funzionalità che il sistema mette a disposizione degli utenti: queste sono suddivise in base al modulo cui fanno riferimento e l'accesso è limitato in base al tipo di account e ai privilegi posseduti.

Nei capitoli che seguono sono elencati i moduli di cui si compone il sistema con i relativi requisiti utente.

2.1.2.1 Modulo di gestione degli utenti

Il presente modulo permette di amministrare qualsiasi aspetto degli account degli utenti attivi nel sistema, oltre a permetterne la creazione. L'accesso a tale modulo deve essere regolamentato onde evitare problematiche di sicurezza. Di seguito il modulo in questione sarà identificato con l'acronimo MGU.

UR1. L'accesso al MGU deve essere ristretto agli utenti SA e AM.

UR2. L'accesso al MGU avviene specificando username e password di un utente.

UR3. Deve essere possibile creare nuovi utenti indicandone le generalità oltre che le credenziali per l'accesso al sistema e la tipologia di utente (AM oppure OP).

UR4. Deve essere possibile disabilitare l'account di un utente, al fine di impedirne l'accesso per motivi amministrativi.

UR5. Deve essere possibile modificare le generalità di un utente (AM oppure OP).

UR6. In ogni momento deve esistere sempre almeno un AM.

UR7. Il MGU deve mostrare un elenco degli utenti accreditati nel sistema.

UR8. Per ogni utente del sistema deve essere possibile visualizzarne le generalità e i privilegi concessi.

UR9. Tutte le operazioni effettuate da un utente devono essere sempre consultabili, anche nel caso in cui l'account utente venga disabilitato.

Gli utenti possono essere raggruppati: lo scopo della creazione di gruppi non è quella di consentire l'accesso a specifiche funzioni del sistema, bensì quello di snellire il processo di notifica di messaggi o allarmi a più utenti contemporaneamente.

UR10. Gli utenti SA e AM devono poter creare dei nuovi gruppi, specificandone il nome e gli utenti appartenenti (AM oppure OP).

UR11. Possono esistere gruppi senza utenti.

UR12. Gli utenti SA e AM devono poter eliminare un gruppo, senza che gli utenti collegati vengano rimossi.

UR13. Gli utenti SA e AM devono poter modificare l'elenco degli utenti associati ad un gruppo.

UR14. Gli utenti SA e AM devono poter visualizzare l'elenco dei gruppi creati.

UR15. Gli utenti SA e AM devono poter visualizzare l'elenco degli utenti associati ad un gruppo.

Trattandosi di un sistema condiviso ed essendo mission critical per una organizzazione che lo utilizza, è opportuno implementare politiche di sicurezza facendo sì che qualsiasi operazione effettuata da un utente sia registrata.

UR16. Qualsiasi operazione effettuata da un utente, compreso l'accesso e l'uscita dal sistema, deve essere registrata, indicando anche le relative modifiche effettuate.

UR17. I dati relativi agli utenti devono essere protetti e inaccessibili da utenti malintenzionati.

2.1.2.2 Modulo di gestione dei privilegi

Il presente modulo permette di concedere ad un utente l'accesso a determinate funzionalità del sistema, oltre a permettere la definizione di nuovi privilegi per lo sviluppo di nuove funzionalità. Di seguito il modulo in questione sarà identificato con l'acronimo MGP.

UR18. Gli utenti SA e AM devono poter concedere privilegi ad altri utenti per permettere loro l'accesso alle funzionalità del sistema.

UR19. Gli utenti SA e AM devono poter revocare i privilegi ad altri utenti per negare loro l'accesso.

UR20. Quando si realizzano nuove funzionalità, il sistema deve permettere di limitarne l'accesso utilizzando il meccanismo dei privilegi.

UR21. L'utente SA deve poter visualizzare un elenco dei privilegi che l'utente SV può utilizzare durante lo sviluppo per limitare l'accesso alle funzionalità del sistema.

2.1.2.3 Modulo di gestione dei dispositivi

Si tratta del modulo più importante del sistema, attraverso il quale è possibile gestire tutti i dispositivi da cui si intendono ricevere i messaggi di stato. Di seguito il modulo in questione sarà identificato con l'acronimo MGD.

UR22. Gli utenti SA, AM e OP con privilegio, devono poter visualizzare l'elenco dei dispositivi inseriti nel sistema.

UR23. Gli utenti SA, AM e OP con privilegio, devono poter inserire un nuovo dispositivo specificandone i dati tecnici ed identificativi, attraverso una procedura guidata.

UR24. Gli utenti SA, AM e OP con privilegio, devono poter disabilitare un dispositivo, mantenendo tutto lo storico relativo.

UR25. Gli utenti SA, AM e OP con privilegio, devono poter modificare i dati tecnici ed identificativi di un dispositivo, in qualsiasi momento.

UR26. Per ogni terminale, l'utente SA, AM oppure un OP con privilegio, devono poter visualizzare le informazioni che lo caratterizzano.

UR27. Specificando l'identificativo di un dispositivo, deve essere possibile localizzarlo entro la struttura organizzativa in cui opera.

UR28. Le informazioni di stato inviate dai dispositivi devono essere elaborate nel più breve tempo possibile.

UR29. Le informazioni di stato inviate dai dispositivi devono essere memorizzate e rese disponibili per effettuare elaborazioni successive o estrapolare dati statistici.

UR30. Quando le informazioni ricevute dai dispositivi sono tali da ritenere necessario informare uno o più utenti (anche indicando un gruppo specifico), il sistema deve mettere a disposizione dell'utente SV alcune funzionalità per l'invio di messaggi agli utenti.

Al fine di adattarsi allo specifico contesto organizzativo in cui il sistema si troverà ad operare, occorre che sia possibile specificare la struttura organizzativa in cui è organizzata l'azienda che acquista la piattaforma. Tale struttura viene utilizzata per localizzare i dispositivi.

UR31. L'utente SA deve poter specificare la struttura del contesto organizzativo in cui il sistema lavora.

UR32. L'utente SA deve poter modificare la struttura del contesto organizzativo in cui il sistema lavora.

2.1.2.4 Modulo di gestione delle attività programmate

Questo modulo si occupa di mandare in esecuzione dei task specifici, creati dallo SV, per eseguire specifici compiti. Di seguito il modulo in questione sarà riferito con l'acronimo MGAP.

UR33. Il sistema deve analizzare i log dei dispositivi ad intervalli regolari ravvicinati (ad esempio potrebbe essere 1 minuto).

UR34. L'accesso al modulo MGAP deve essere riservato all'utente SA.

UR35. Il modulo MGAP deve permettere di inserire un nuovo task nell'ordine che preferisce, specificandone la cadenza di esecuzione.

UR36. Il modulo MGAP deve permettere di rimuovere un task.

UR37. Il modulo MGAP deve permettere di modificare l'ordine di esecuzione dei task.

2.2 Analisi e specifica dei requisiti di sistema

Definite le necessità in termini di funzionalità da realizzare, occorre focalizzare l'attenzione sui vincoli da imporre per realizzare il sistema nel pieno rispetto dei requisiti iniziali: di seguito sono stati definiti i requisiti che il sistema dovrà soddisfare sulla base dei requisiti espressi dall'utente nel URD (vedi capitolo 2.1).

Il documento di specifica dei requisiti di sistema, o SRD, contiene le caratteristiche da rispettare durante lo sviluppo e servirà da base di verifica, durante il collaudo, per verificare il rispetto dei requisiti inizialmente espressi dal cliente.

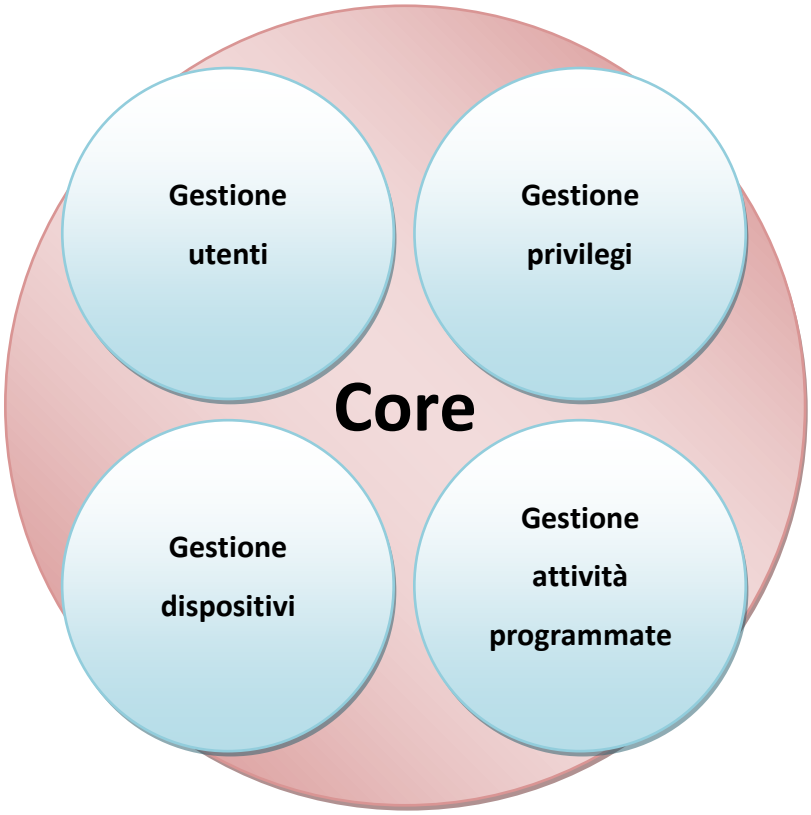


Figura 4 – Composizione per moduli della piattaforma

I requisiti di sistema sono esposti in maniera misurabile e verificabile, esplicitando il più possibile le necessità di realizzazione, trattandosi di un applicativo sviluppato per essere poi personalizzato per un certo cliente.

Per ogni requisito di sistema saranno indicati, tra parentesi quadre, i requisiti utente che questo soddisfa. Tuttavia possono esistere requisiti di sistema che non soddisfano alcun requisito utente, ma tutti i requisiti utente devono essere soddisfatti da almeno un requisito di sistema.

Nel presente documento, saranno presenti tecnicismi ed accenni di soluzioni implementative, così da fornire agli utenti SV la possibilità di personalizzare il comportamento dell'aplicativo.

SR1. Il sistema, compresi i moduli nei capitoli che seguono, non deve essere modificato in alcun modo durante la personalizzazione.

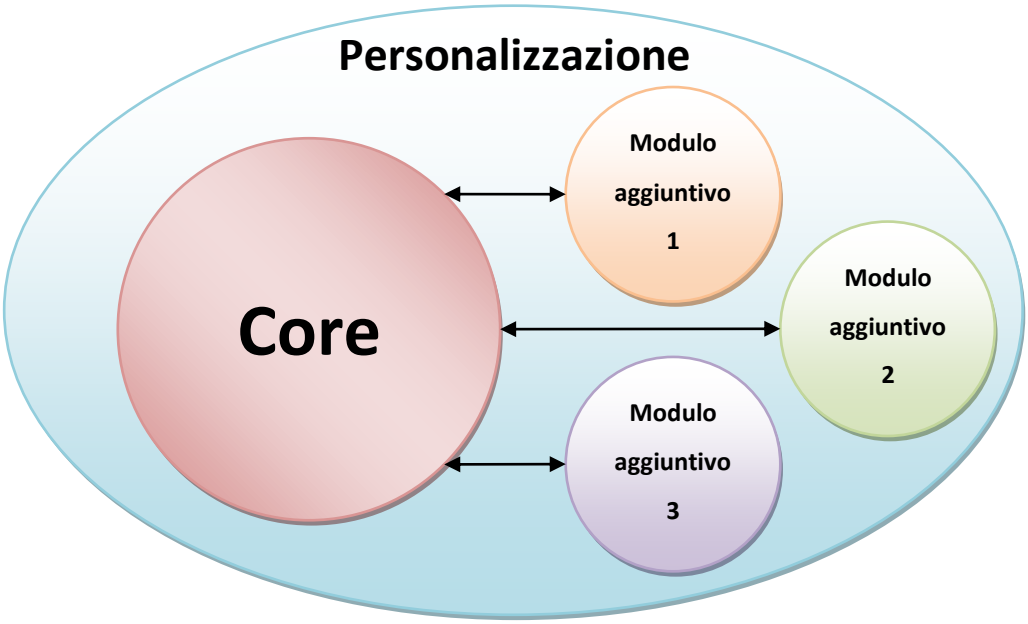


Figura 5 – Esempio di personalizzazione della piattaforma per un utilizzo specifico

2.2.1 Core di sistema

Tutte le funzionalità a livello di codice che devono essere rese disponibili a colui che sviluppa la personalizzazione della piattaforma sono contenute nel core di sistema. Di seguito il core sarà indicato con la sigla CS.

SR2. Deve essere presente una funzione che permetta l'invio di un messaggio ad un utente o ad un gruppo di utenti. [UR30]

SR3. Deve essere possibile memorizzare variabili di stato a livello globale.

SR4. Deve essere possibile leggere le variabili di stato globali.

2.2.2 Modulo di gestione degli utenti

Il MGU dovrà essere sviluppato al solo scopo di permettere la gestione degli account utente. Data la criticità dello strumento, l'accesso allo stesso dovrà essere limitato a quei soggetti che hanno l'autorità di concedere l'accesso al sistema.

SR5. L'accesso al MGU deve essere riservato agli utenti SA e AM. [UR1]

Data la generalità della piattaforma, in una installazione della stessa, potrebbe essere necessario specificare alcuni campi che descrivono un utente (es. nome,

cognome, indirizzo di posta elettronica), mentre in un'altra installazione potrebbero essere differenti (es. nome, cognome, sesso). Questo implica che il sistema debba essere configurabile sotto questo punto di vista, per adattarsi alle necessità di ogni contesto organizzativo.

SR6. I campi che descrivono un utente devono poter essere definiti da parte del SA.

SR7. Le informazioni minime per descrivere un utente sono:

- username;
- password;
- password salt;
- indirizzo di posta elettronica.

L'interfaccia di gestione degli utenti deve fornire all'utente autorizzato le funzionalità tipiche di creazione, modifica e disabilitazione di un utente. La rimozione non viene contemplata in quanto l'eliminazione di un utente comporterebbe la perdita dello storico delle operazioni eseguite da esso.

SR8. Il MGU deve permettere la creazione di nuovi utenti specificando i valori per i campi che descrivono un utente, definiti con la funzionalità del requisito SR7. [UR3]

SR9. Il MGU deve permettere la modifica di campi che descrivono un utente. [UR5]

SR10. Il MGU deve permettere di disabilitare l'account di un utente, senza eliminarlo. [UR4, UR9]

SR11. Non è possibile disabilitare l'account dell'utente SA.

SR12. Il MGU deve permettere di abilitare un account utente che era stato disabilitato.

SR13. Un utente, utilizzando username e password, accede al sistema, qualora il suo account non risulti disabilitato. [UR2, UR4]

SR14. Un utente non può disabilitare il proprio account. [UR6]

SR15. Il MGU deve prevedere una interfaccia per la visualizzazione di tutti gli utenti registrati nel sistema, compresi quelli disabilitati, evidenziandone la condizione di utente abilitato/disabilitato e il tipo di account (AM oppure OP). [UR7]

SR16. L'interfaccia di visualizzazione degli utenti accreditati non deve mostrare l'account del SA.

SR17. Deve essere possibile visualizzare tutte le informazioni memorizzate di un utente selezionato. [UR8]

SR18. La creazione di un nuovo utente è un'operazione che deve essere registrata nel log. [UR16]

SR19. La modifica delle informazioni di un utente esistente è un'operazione che deve essere registrata nel log. [UR16]

SR20. La disabilitazione di un utente esistente è un'operazione che deve essere registrata nel log. [UR16]

SR21. L'accesso di un utente con le proprie credenziali è un'operazione che deve essere registrata nel log. [UR16]

Gli utenti devono poter essere raggruppati per facilitare l'identificazione di persone che svolgono le medesime funzioni. A tale scopo i gruppi permettono di raccogliere questi utenti per identificarli rapidamente tramite un identificativo univoco di gruppo. Eventuali messaggi di notifica o di allarme possono essere inviati a molti soggetti, specificandone l'identificativo di gruppo.

SR22. Il MGU deve permettere la creazione di un nuovo gruppo, indicandone il nome e gli eventuali utenti appartenenti. [UR10, UR11]

SR23. Il MGU deve permettere di rimuovere un gruppo esistente, disassociando gli utenti collegati allo stesso. [UR12]

SR24. Il MGU deve permettere di aggiungere utenti ad un gruppo. [UR13]

SR25. Il MGU deve permettere di rimuovere utenti da un gruppo. [UR13]

SR26. L'interfaccia deve permettere di visualizzare tutti i gruppi disponibili nel sistema, indicando gli utenti che vi appartengono. [UR14, UR15]

SR27. La creazione di un nuovo gruppo è un'operazione che deve essere registrata nel log. [UR16]

SR28. La modifica dei componenti di un gruppo è un'operazione che deve essere registrata nel log. [UR16]

SR29. La rimozione di un gruppo esistente è un'operazione che deve essere registrata nel log. [UR16]

Una volta che un utente ha avuto accesso al sistema, occorre prevedere una pagina in cui egli possa gestire il proprio profilo sul sistema.

SR30. L'utente che ha avuto accesso al sistema deve poter modificare la propria password.

SR31. La modifica della password personale da parte di un utente deve essere registrata nel log. [UR16]

2.2.3 Modulo di gestione dei privilegi

Il modulo che andiamo a descrivere svolge una duplice funzione nell'ambito dell'amministrazione e gestione della piattaforma: da un lato permette di concedere o rimuovere permessi agli utenti al fine di regolarne l'accesso alle funzioni disponibili, mentre dall'altro lato permette di integrare nuove funzionalità nel sistema personalizzato.

Esistono due tipologie di privilegi:

1. **privilegio predefinito**: si tratta di un privilegio che non può essere modificato o eliminato, nemmeno dal Super Amministratore; questo è impostato in modo predefinito ed è parte integrante del software;
2. **privilegio personalizzato**: allo scopo di limitare l'accesso a determinate funzionalità del sistema, è possibile definire un privilegio personalizzato da utilizzare, da parte dello sviluppatore, quando questi realizza una funzione specifica dell'interfaccia di amministrazione del sistema.

I comandi forniti nel MGP per l'impostazione dei privilegi definiscono quali utenti possono effettuare quali operazioni nel sistema.

SR32. L'accesso al MGP deve essere riservato al SA e agli utenti AM. [UR18, UR19]

SR33. Il MGP deve permettere la concessione di un privilegio esistente ad un utente. [UR18]

SR34. Il MGP deve permettere di revocare un privilegio ad un utente. [UR19]

SR35. Il MGP deve permettere di visualizzare l'elenco di tutti i privilegi personalizzati disponibili nel sistema. [UR21]

SR36. La concessione di un privilegio ad un utente è un'operazione che deve essere registrata nel log. [UR16]

SR37. La revoca di un privilegio ad un utente è un'operazione che deve essere registrata nel log. [UR16]

Per permettere la personalizzazione del sistema, secondo le specifiche esigenze del cliente finale, è necessario che il MGP metta a disposizione dello sviluppatore alcune funzioni che gli permettono di indicare alla piattaforma la creazione di nuovi privilegi. L'accesso alle funzioni personalizzate è vincolato al fatto che un utente possegga il privilegio necessario.

La definizione di un nuovo privilegio di sistema implica l'assegnamento allo stesso di un codice identificativo univoco con cui lo SV limiterà le funzionalità del sistema.

SR38. Il MGP deve permettere solo al SA di creare nuovi privilegi personalizzati, fornendone il nome descrittivo.

SR39. Alla creazione di un nuovo privilegio personalizzato, il sistema deve restituire un identificativo univoco tra tutti i privilegi. [UR20]

SR40. Gli attributi descrittivi di un privilegio esistente non devono poter essere modificati.

SR41. Un privilegio esistente non deve poter essere eliminato.

SR42. La creazione di un privilegio personalizzato è un'operazione che deve essere registrata nel log. [UR16]

2.2.4 Modulo di gestione dei dispositivi

L'obiettivo della piattaforma è la gestione dei messaggi di stato dei dispositivi embedded appartenenti ad una azienda che utilizza la piattaforma. I dispositivi, cellulari o altri che siano, ricoprono quindi un ruolo di primaria importanza, essendo questi che alimentano il sistema.

Figura 6 - Esempio di contesto organizzativo di una azienda

Anche questo modulo, come il MGU, svolge una doppia funzione:

1) permette l'amministrazione dei dispositivi embedded da cui il sistema riceve i messaggi di stato;
2) permette di indicare la struttura organizzativa in cui si articola l'azienda che acquista la piattaforma.

La prima funzione viene solitamente svolta da un soggetto autorizzato che possiede i privilegi relativi: ad esempio, potrebbe dover essere necessario consentire l'inserimento di un nuovo dispositivo ad un soggetto OP, invece che riservarne il compito all'amministratore.

SR43. L’utente SA, l’utente AM o un utente OP con privilegio, devono poter visualizzare la lista completa di tutti i dispositivi registrati nel sistema. [UR22]

SR44. L’utente SA, l’utente AM o un utente OP con privilegio, devono poter inserire un nuovo dispositivo, specificando le informazioni che lo descrivono, in base al tipo di dispositivo. [UR23]

SR45. L’utente SA, l’utente AM o un utente OP con privilegio, devono poter disabilitare un dispositivo esistente. [UR24]

SR46. L’utente SA, l’utente AM o un utente OP con privilegio, devono poter abilitare un dispositivo disabilitato.

SR47. L’utente SA, l’utente AM o un utente OP con privilegio, devono poter modificare i dati di un dispositivo in qualsiasi momento. [UR25]

SR48. L’utente che può visualizzare la lista dei dispositivi di sistema, deve poter visualizzare le caratteristiche di un dispositivo scelto. [UR26]

SR49. Specificando l'identificativo di un dispositivo, il sistema deve restituire la sua localizzazione entro il contesto organizzativo. [UR27]

SR50. L'inserimento di un nuovo dispositivo è un’operazione che deve essere registrata nel log. [UR16]

SR51. La modifica dei dati di un dispositivo è un’operazione che deve essere registrata nel log. [UR16]

SR52. La disabilitazione di un dispositivo è un’operazione che deve essere registrata nel log. [UR16]

Data la generalità della piattaforma, in una organizzazione potrebbero esistere dispositivi con caratteristiche diverse (es. uno smartphone ha caratteristiche diverse da un POS). Questo implica che debbano poter essere definiti dei modelli di dispositivi, con le relative caratteristiche, per descrivere tutte le tipologie di dispositivo presenti.

SR53. I campi che descrivono un tipo di dispositivo devono poter essere definiti da parte del SA.

SR54. I campi sempre presenti che descrivono un dispositivo sono:

- identificativo univoco;
- posizione entro la struttura organizzativa.

Ad esempio potremmo definire i seguenti tipi di dispositivo:

Modello Nokia N97	Modello POS Rx35
• Numero seriale	• Numero seriale
• Buffer interno	• Memoria di sistema
• Memory card	• Produttore

Tabella 1 - Esempio di tipi di dispositivo

Riprendendo le funzioni che questo modulo svolge, la seconda funzione, svolta esclusivamente dal SA e dagli AM, permette di specificare la struttura organizzativa (vedi Figura 7), per localizzare i dispositivi.

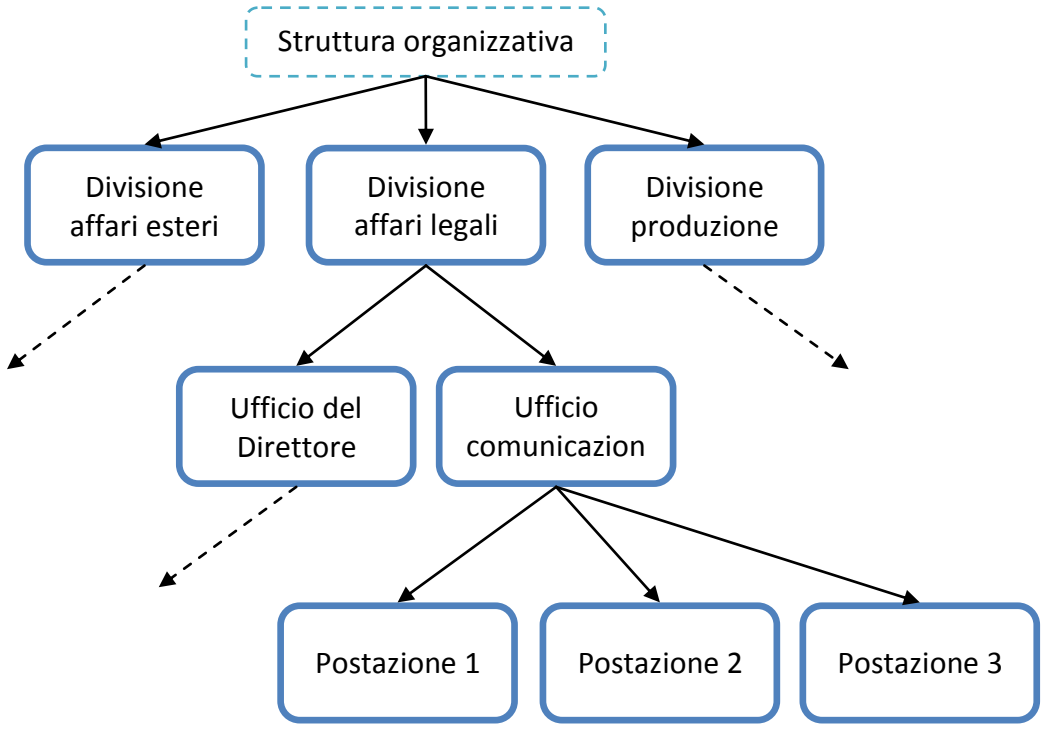


Figura 7 - Esempio di definizione di una struttura organizzativa

SR55. L'utente SA deve poter specificare la struttura organizzativa. [UR31, UR32]

SR56. La modifica della struttura organizzativa può avvenire in qualsiasi momento, rispettando le politiche di consistenza dei dati.

SR57. Un nodo della struttura organizzativa non deve poter essere modificato qualora esistano dispositivi associati allo stesso nodo o ad uno dei suoi discendenti.

Consideriamo la Figura 7: non dovrei poter modificare alcuna informazione della postazione 1, né dell'ufficio comunicazione, né della divisione affari legali, mentre dovrei poterlo fare per la postazione 2 e la postazione 3 (sempre se non avessero dispositivi associati).

SR58. Un dispositivo deve poter essere localizzato anche su nodi non foglia della struttura organizzativa reale.

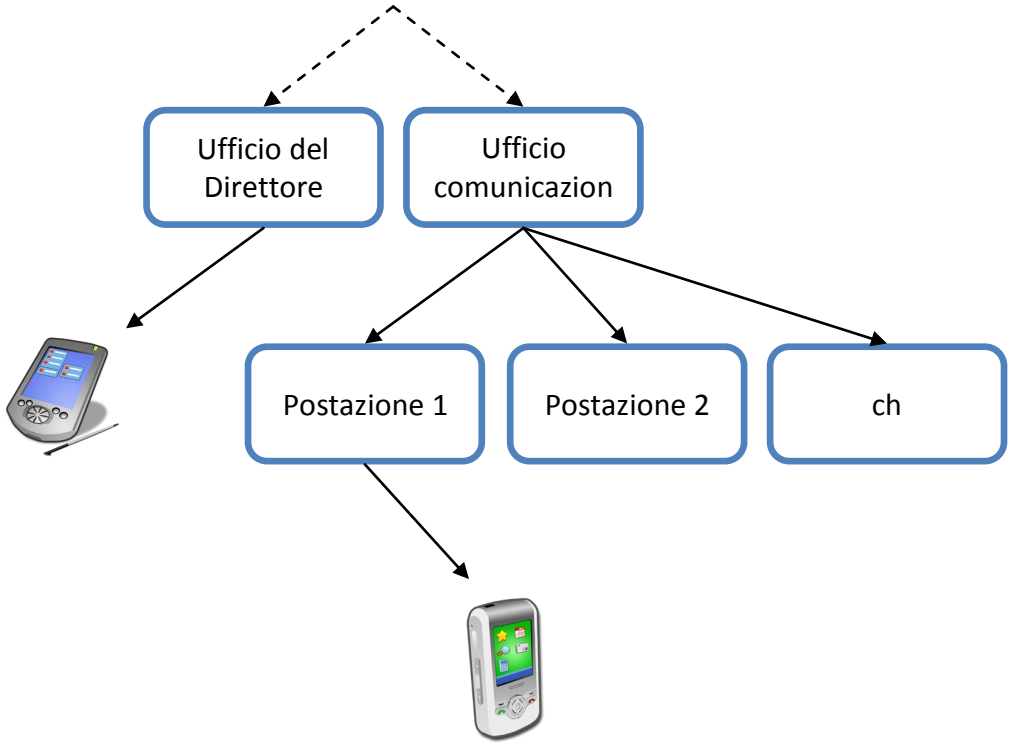


Figura 8 - Esempio di localizzazione dei dispositivi

2.2.5 Modulo di gestione delle attività programmate

Al fine di alimentare il sistema con i messaggi di stato dei dispositivi, occorre che i messaggi stessi siano disponibili il prima possibile e siano interpretati correttamente. Data la natura eterogenea dei dispositivi (vedi capitolo 2.2.4), occorre fornire un meccanismo dinamico di interpretazione che si adatti a qualsiasi necessità dello sviluppatore: infatti, come abbiamo visto precedentemente, il core del sistema non deve essere in alcun modo modificato, quindi esso stesso dovrà essere sufficientemente generico da supportare qualsiasi tipologia di dato.

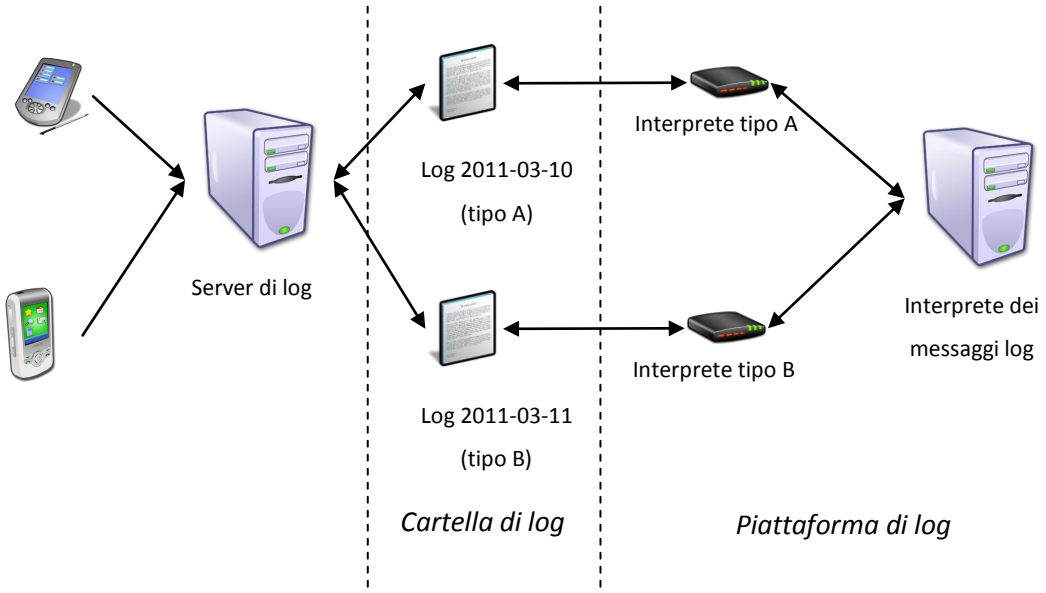


Figura 9 - Schema di funzionamento del sistema

SR59. L'accesso al MGAP deve essere ristretto all'utente SA. [UR34]

SR60. La piattaforma reperisce i file log da manipolare, all'interno di una cartella, denominata "cartella di log".

SR61. L'utente SA deve poter modificare il percorso della "cartella di log".

Per mantenere aggiornato il sistema, la lettura della "cartella di log" deve essere effettuata con una alta frequenza.

SR62. La "cartella di log" deve essere analizzata a intervalli di 1 minuto. [UR28, UR33]

Come si nota facilmente dalla Figura 9, possono esistere diversi formati con cui i log vengono scritti dal server di log. Il sistema non può prevedere tutti i possibili formati esistenti, quindi occorre che lo stesso fornisca allo SV di una personalizzazione la possibilità di far comprendere al sistema un certo formato: tale necessità è soddisfatta con l'uso degli interpreti. Ognuno di questi è in grado di comprendere il formato di un certo log e salvare nel sistema le informazioni in modo coerente. Sta quindi allo SV realizzare lo specifico interprete per un certo formato di log. Parliamo di "formato di log" in quanto risulta chiaro che differenti dispositivi possono comunicare i loro messaggi nel medesimo formato.

SR63. Il sistema deve permettere allo SV di realizzare ed installare un nuovo interprete sulla piattaforma.

SR64. Il sistema deve permettere allo SV di modificare un interprete esistente.

SR65. Il sistema deve permettere allo SV di rimuovere un interprete esistente.

Durante l'interpretazione di un certo file di log, l'interprete può accorgersi di situazioni particolari che richiedono un'azione: ad esempio potrebbe essere necessario inviare un messaggio posta elettronica di avviso quando si verificano un certo numero di eventi.

SR66. Il sistema deve mettere a disposizione dello SV di un interprete, una struttura dati in cui poter memorizzare informazioni di stato tra una esecuzione e l'altra dello stesso interprete. [UR29, SR3, SR4]

Quanto espresso nel precedente capoverso è relativo ad azioni scatenate in conseguenza della lettura di un file di log. Potrebbe essere necessario svolgere operazioni (task) anche quando non esistono file di log da leggere, e comunque ad orari prestabiliti: si configura la necessità di disporre, a livello di sistema, di un gestore per le attività programmate (cron).

SR67. Deve essere possibile inserire un nuovo task, indicandone: [UR35]

- file da eseguire;
- cadenza di esecuzione (con granularità di minuto).

SR68. Deve essere possibile modificare i dati relativi ad un task, compreso l'ordine di esecuzione. [UR35, UR37]

SR69. Deve essere possibile rimuovere un task. [UR36]

SR70. Deve essere possibile disabilitare un task.

SR71. Deve essere possibile abilitare un task.

SR72. Il sistema, ad intervalli regolari (così come indicato nel requisito SR62), deve mandare in esecuzione prima tutti gli interpreti e poi verificare se ci sono task da eseguire.

Al fine di mantenere sempre aggiornato il core di sistema con eventuali bug fix e miglioramenti, si rende necessario prevedere un meccanismo di sincronizzazione con gli sviluppatori della piattaforma.

SR73. Il sistema deve prevedere la sincronizzazione del codice della piattaforma con il server di aggiornamento, almeno una volta al giorno.

2.2.6 Modulo di log delle attività

Per garantire la sicurezza del sistema si rende necessario monitorare tutte le attività effettuate dai vari utenti. Il modulo in questione, da qui in poi denominato MLA, si occupa proprio di registrare qualsiasi operazione venga effettuata dall'utente collegato.

SR74. Le operazioni degli utenti che implichino una modifica dei dati contenuti nel sistema devono essere registrate. [UR16]

SR75. L'accesso al MLA deve essere riservato all'utente SA e AM.

SR76. Deve essere possibile visualizzare l'elenco delle operazioni effettuate da un certo utente.

SR77. Deve essere possibile filtrare le operazioni effettuate in base ad un intervallo temporale.

2.2.7 Tecnologie utilizzate in fase di implementazione

2.2.7.1 Tecnologie lato client

Gli utilizzatori del sistema potrebbero avere a disposizione mezzi molto differenti per composizione hardware ed architettura software (sistema operativo e browser): per tale motivo la fase di test dovrà avvenire su diverse piattaforme.

Le pagine Web dovranno essere visualizzate correttamente con i Web Browser associati ai sistemi operativi maggiormente diffusi (Windows, Unix/Linux, MacOS). L'elenco aggiornato con le statistiche d'uso utilizzato per redigere i seguenti punti è presente in Tabella 2.

A tal fine è stata utilizzata una combinazione di linguaggi standard per implementare le diverse parti del sito Web.

I linguaggi utilizzati per la realizzazione del portale dovranno essere:

- XHTML v1.1: struttura della pagine;
- CSS v2.0: fogli di stile;
- jQuery v1.6: programmazione di effetti ed interazioni con l'utente.

Per la programmazione lato Client è stata scelta la libreria jQuery che permette di fare animazioni, manipolare elementi HTML, gestire eventi e fare richieste AJAX con poche righe di codice, senza preoccuparsi della particolare versione di JavaScript in uso. Tale libreria dispone innumerevoli plugin che la rendono uno degli strumenti più flessibili e dinamici per sviluppare GUI all'avanguardia e usabili.

2.2.7.1.1 XHTML 1.1

Il linguaggio di scrittura della pagine web è l'HTML: negli ultimi anni questo ha visto notevoli cambiamenti e miglioramenti alla sua struttura e semantica dei tag, arrivando alla definizione del XHTML, un linguaggio di marcatura che associa alcune proprietà dell'XML con le caratteristiche dell'HTML.

Il linguaggio prevede un uso più restrittivo dei tag HTML sia in termini di validità che in termini di sintassi per descrivere solo la struttura logica della pagina, mentre il

layout e la resa grafica sono imposti dai fogli di stile a cascata (Cascading Style Sheets, CSS).

Esistono varie versioni di XHTML 1.0 e ad ognuna di esse è associata una specifica Document Type Definition (DTD) che definisce l'insieme di regole mediante le quali un dato documento può essere renderizzato (cioè rappresentato correttamente) dall'XHTML.

- **XHTML 1.0 Transitional**: nato per favorire la migrazione dalla vecchia versione HTML 3.2 o per uso insieme a link e frame in-line. Accetta come validi anche i tag HTML che sono stati definiti come deprecati in XHTML ed è tollerante rispetto ad alcune non conformità sintattiche;
- **XHTML 1.0 Strict**: rispetto alla versione Transitional non accetta i tag HTML definiti deprecati, non è tollerante a non conformità sintattiche e prevede controlli più rigorosi anche rispetto al valore di alcuni attributi dei tag (per esempio, l'attributo id deve avere valori univoci all'interno dello stesso documento);
- **XHTML 1.0 Frameset**: nato per motivi di compatibilità per suddividere la finestra visualizzata dal browser in diversi frame (sottofinestre), pratica un tempo diffusa ma ora deprecata dal World Wide Web Consortium;
- **XHTML 1.0 Loose**: una versione ampiamente tollerante alle incongruenze, permette l'uso di tag deprecati;
- **XHTML 1.1**: si tratta di una riformulazione del XHTML 1.0 Strict ed è disponibile in una sola versione.

La versione che utilizzeremo nello sviluppo del sistema è XHTML 1.1 al fine di garantire la compatibilità con il maggior numero di browser ed evitare incongruenze di visualizzazione tra dispositivi diversi.

2.2.7.1.2 CSS 2.0

Riguardo all'utilizzo del Cascading Style Sheets, particolare attenzione è stata posta sulla scelta degli attributi da utilizzare e soprattutto su quelli da evitare

attentamente. Ad esempio è stato evitato l'uso della proprietà "position: static", in quanto non supportati da uno o più browser. A causa di limitazioni come questa il Web Designer è costretto a ingegnerizzare lo stile del sito Web con un set di parametri ridotto, richiedendo maggiore fantasia in alcune scelte al fine di riuscire comunque ad ottenere l'effetto grafico desiderato.

L'introduzione del CSS v.3 dovrebbe dotare il Web Designer di strumenti più efficaci per la propria creazione grafica. Allo stesso tempo la progressiva scomparsa delle versioni più obsolete dei browser renderà più semplice la codifica di uno stile con una resa non degradata con i diversi browser. Ad esempio Internet Explorer v.8.0 e Mozilla Firefox v.3.0 prevedono l'autoscaling delle immagini, effetto che con le precedenti versioni deve essere sviluppato con un sofisticato artificio in Javascript.

Nella realizzazione del sistema faremo uso dei fogli di stile versione 2.0 perché attualmente i più supportati dai browser in uso (vedi Tabella 2).

2.2.7.1.3 jQuery

jQuery è una libreria JavaScript che permette la manipolazione degli oggetti DOM in modo facile e leggero, oltre a fornire innumerevoli effetti grafici per rendere le interfacce con l'utente usabili quanto un applicativo desktop. Si tratta di un frame work sviluppato da John Resig, già dal 2006, con il preciso intento di rendere il codice più sintetico e di limitare al minimo l'estensione degli oggetti globali per ottenere la massima compatibilità con altre librerie. Tale libreria è in grado di offrire un'ampia gamma di funzionalità, che vanno dalla manipolazione degli stili CSS e degli elementi HTML, agli effetti grafici per passare a comodi metodi per chiamate AJAX cross-browser. Il tutto, appunto, senza toccare nessuno degli oggetti nativi JavaScript. Come detto, infatti, il punto focale del progetto jQuery è il fatto di essere indipendente dal particolare browser utilizzato, fornendo funzioni che nascondono la specifica implementazione delle stesse rispetto alla versione di JavaScript in uso.

Uno dei punti di forza di tale progetto è la brevità del codice utilizzato per realizzare azioni complesse (il motto ufficiale infatti è "Write Less Do More"), che si fonda sul potente motore di selezione degli oggetti sotto forma di selettori CSS.

Il motore di jQuery è l'oggetto/funzione $ che permette la selezione rapida di qualsiasi oggetto presente sul DOM, anche quelli aggiunti dinamicamente. Ad esempio per selezionare tutti gli oggetti che hanno la classe "miaClasse" basterà scrivere:

```
$(".miaClasse");
```

2.2.7.2 Tecnologie lato server

2.2.7.2.1 PHP 5.3

Il linguaggio server-side scelto per la realizzazione della parte logica del sistema è PHP, nello specifico la versione 5.3 o successive. Questo linguaggio ha come obiettivo principale quello di sviluppare applicazioni web lato server, ma può essere utilizzato anche per realizzare script a riga di comando o applicazioni stand-alone con interfaccia grafica.

PHP (acronimo ricorsivo di “PHP: Hypertext Preprocessor”, preprocessore di ipertesti) è un linguaggio di scripting interpretato ampiamente diffuso e costantemente sviluppato, con licenza open source e libera: per questi motivi risulta un’ottima scelta per la tecnologia lato server.

Data la necessità di utilizzare particolari istruzioni, al fine di semplificare la scrittura del codice, il sistema deve essere sviluppato utilizzando almeno la versione 5.3 di PHP.

SR78. Il sistema deve utilizzare il linguaggio di scripting server side PHP in versione 5.3 o superiore.

2.2.7.2.2 MySQL

Il RDBMS MySQL 5 è stato scelto perché assai diffuso, open-source, compatibile con le architetture software più diffuse e con i linguaggi di programmazione più utilizzati. MySQL si ritrova nei pacchetti WAMP e LAMP che includono Apache e PHP oltre che a questo DBMS.

MySQL permette l'utilizzo di DB transazionali attraverso storage engine come InnoDB e Falcon. Il motore InnoDB è quello scelto per la memorizzazione delle tabelle e dei record per questo tipo di applicazione poiché permette:

- l'uso delle transazioni;
- lock a livello di record;
- chiavi esterne e vincoli di integrità referenziale.

Nel capitolo 3.3.2 sono elencate le funzionalità ricercate ed è possibile comprendere le motivazioni per cui è stato scelto questo motore invece di quello predefinito MyISAM.

2.2.7.2.3 Apache

Il Web Server Apache pemette ad ogni richiesta del client la suddivisione in moduli delle funzionalità specifiche come unità indipendenti. I moduli sono gestiti in modo sequenziale e la sequenza è inizializzata da un modulo denominato "core". Apache permette di gestire richieste sia in chiaro (http) sulla porta 80 che in modalità sicura (ssl) sulla porta 443.

Apache viene configurato a partire dal file *httpd.conf* e offre la possibilità di caricare e gestire moduli aggiuntivi.

2.2.7.3 Design pattern Model-View-Controller

Data la necessità che il sistema sia visualizzabile anche su dispositivi mobili, si rende necessario realizzare il sistema in modo tale da presentare una pagina in modo diverso rispetto al browser che la richiede. Il tipico approccio a questo tipo di problema è il design pattern Model-View-Controller: con questo metodo di realizzazione la realizzazione del sistema viene suddivisa in tre fasi:

1. **Realizzazione del modello (***model***)**: fornisce i metodi per accedere ai dati utili all'applicazione;
2. **Progettazione della logica di business (***controller***)**: riceve i comandi dell'utente (in genere attraverso la vista) e li esegue modificando lo stato

degli altri due componenti (ad esempio modificando l'interfaccia grafica e modificando il database);

3. **Realizzazione della vista (*view*)**: visualizza i dati contenuti nel modello in base al tipo di client che effettua la richiesta e gestisce l'interazione con l'utente.

Si rende quindi necessario disporre di questo tipo di pattern architetturale.

SR79. In fase di progettazione occorre prevedere la separazione della logica dalla grafica, ossia poter disporre di diversi modelli grafici di presentazione sulla base delle specifiche tecnologie utilizzate dal client con il quale si accede al portale (ad esempio utilizzare template manager come Smarty per PHP).

2.2.7.3.1 Model-View-Controller: Smarty

Per applicare il Model-View-Controller al sistema utilizzeremo Smarty, un motore di web template scritto in PHP. Smarty consente di separare il codice PHP, il business logic (la programmazione del software), dal codice HTML, il presentation logic (l'aspetto grafico di un sito web), e di generare contenuti web mediante il rimpiazzo di speciali Smarty tag all'interno del documento (sostituzione di variabili, etc.).

Un tag è una direttiva, racchiusa da speciali caratteri (tipicamente parentesi graffe), interpretata dal motore di Smarty, che possono rappresentare variabili, denotate dal simbolo del dollaro ($), funzioni, o anche istruzioni di controllo del flusso. Smarty permette ai programmatori PHP di definire delle funzioni da includere nei tag stessi di Smarty.

La raffinata astrazione creata da Smarty consente di separare in maniera netta l'interfaccia grafica di una pagina web (la presentazione) dal back-end in PHP, favorendo lo sviluppo di applicazioni di gran lunga più organizzate ed eleganti: è infatti possibile "nascondere" ai Web Designer la logica di programmazione di un sito, mentre i programmatori lavoreranno senza curarsi dell'interfaccia grafica del sito.

L'esempio riportato sul sito ufficiale è un semplice Hello World, suddiviso in 2 file:

- index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//IT"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
<head>
    <title>{$title_text}</title>
    <meta http-equiv="content-type" content="text/html; charset=iso-
8859-1" />
</head>

<body> {* Piccolo commento che NON sarà visibile nel sorgente HTML
*}
    <p>{$body_text}</p>
</body><!-- Piccolo commento che sarà visibile nel sorgente HTML -->
</html>
```

- index.php

```
<?php define('SMARTY_DIR', 'smarty-2.6.9/');
require_once(SMARTY_DIR . 'Smarty.class.php');

$smarty = new Smarty();
$smarty->template_dir = './templates/';
$smarty->compile_dir = './templates/compile/';
$smarty->assign('title_text', 'TITOLO: Un semplice esempio sull\'uso
di Smarty...');
$smarty->assign('body_text', 'BODY: Questo e\' il messaggio
impostato con assign()');

$smarty->display('index.html');
?>
```

In file PHP gestisce la logica dell'applicazione, comunicando al file html (la grafica) quali sono le variabili da inserire. Le funzioni membro dell'oggetto $smarty più importanti sono:

- *display($template_file)*: permette di avviare il motore di Smarty mostrando il file $template_file;
- *assign($nome_var, $local_var)*: permette di far vedere $local_var al motore di Smarty quando compila la grafica col nome specificato in $nome_var.

2.2.8 Usabilità dell'interfaccia

2.2.8.1 Compatibilità con i maggiori browser

Nella realizzazione di un sito web un aspetto che deve essere sicuramente tenuto in considerazione è quello della compatibilità. È necessario che gli utenti che

visualizzano le pagine, anche se utilizzano diversi browser, sistemi operativi o piattaforme hardware, possano fare ciò in modo veloce e sicuro.

Il sistema dovrà essere accessibile e correttamente visualizzabile sui browser più diffusi, sia per dispositivi desktop che per dispositivi mobili. Per questo motivo si cercherà di usare il meno possibile particolari caratteristiche di un browser per realizzare le nostre pagine. La struttura delle pagine HTML potrà quindi essere visualizzata senza il requisito di particolari plug-in, ma semplicemente di alcuni strumenti embedded nei browser elencati nel requisito SR80.

SR80. Il sistema deve essere testato e funzionante con i seguenti Browser:

- Internet Explorer v7.0;
- Internet Explorer v8.0;
- Internet Explorer v9.0;
- Firefox v3.0 o superiore;
- Opera v7.0 o superiore;
- Safari v2.0 o superiore;
- Chrome v10.0 o superiore.

2011	Internet Explorer	Firefox	Chrome	Safari	Opera
Aprile	24.3 %	42.9 %	25.6 %	4.1 %	2.6 %
Marzo	25.8 %	42.2 %	25.0 %	4.0 %	2.5 %
Febbraio	26.5 %	42.4 %	24.1 %	4.1 %	2.5 %
Gennaio	26.6 %	42.8 %	23.8 %	4.0 %	2.5 %

Tabella 2 - Statistiche di utilizzo dei vari browser

Particolare attenzione deve essere posta nello sviluppo del sistema per iPhone, il più diffuso tra i dispositivi mobili.

SR81. Il sistema deve essere funzionante e correttamente visualizzato sul browser dell'IPhone.

2.2.8.2 Facilità di navigazione

2.2.8.2.1 Adattabilità alle diverse risoluzioni

Come indicato precedentemente, nel realizzare la presentazione grafica all'utente si porrà particolare attenzione all'impatto che questa avrà sull'usabilità. Si terrà conto dei diversi contesti d'uso e dei diversi dispositivi di visualizzazione (monitor, cellulari, palmari, etc.).

SR82. La risoluzione minima pienamente supportata su dispositivi desktop è 1024x768 px, considerata la diffusione di monitor a risoluzioni elevate.

SR83. La risoluzione minima pienamente supportata su dispositivi mobili è 240x320 px.

2.2.8.3 Strumenti di feedback e messaggi di errore

Quando un utente compie un'azione è opportuno che questo abbia riscontro visivo circa l'esito della stessa. Ciò è particolarmente importante nel caso di operazioni critiche quali ad esempio la prenotazioni dei prodotti in caso di errore (fallimento di transazioni, malfunzionamenti software e guasti hardware) è opportuno restituire all'utente una descrizione dell'errore occorso.

SR84. Il sistema deve restituire, in ogni caso, l'esito delle richieste inviate dall'utente.

SR85. In caso di errore il messaggio mostrato all'utente deve indicarne la causa senza scendere in dettagli tecnici.

2.2.9 Sicurezza e consistenza del sistema

2.2.9.1 Memorizzazione dei dati sensibili

SR86. Il sistema deve garantire la segretezza dei dati personali.

SR87. Qualunque password dovrà rispettare le seguenti specifiche:

- essere composta da almeno 6 caratteri;
- essere composta da almeno una lettera e un numero, oltre ai caratteri speciali "-" e "_": [a-zA-Z0-9_];

- dovrà essere diversa dal campo username al quale è collegata.

SR88. Le interazioni tra utente e server che prevedono scambio di informazioni personali devono essere effettuate utilizzando una connessione sicura (ad esempio https) con il supporto di algoritmi di cifratura.

SR89. Le password degli utenti non devono essere memorizzate in chiaro nel database, ma applicandovi un algoritmo di hashing (non reversibile) ad esempio MD5 o SHA-1.

SR90. La sessione di lavoro di un utente deve scadere dopo 30 minuti di inattività.

2.2.9.2 Robustezza agli attacchi SQL Injection

Quando si sviluppa la parte logica di una applicazione che sarà resa pubblica, spesso si fa uso diretto dei valori che l'utente inserisce per effettuare delle query nel database: ad esempio durante una ricerca il contenuto dell'input viene utilizzato direttamente in una condizione di richiesta al database. Se tale campo contiene caratteri speciali quali (' – " \ / ... ;), che notoriamente vengono utilizzati per la scrittura dello scheletro della richiesta SQL, un malintenzionato potrebbe abortire l'esecuzione della query in programma ed eseguirne una propria.

SR91. Il sistema deve permettere all'utente di effettuare richieste utilizzando tutti i caratteri a disposizione, ma queste devono essere opportunamente filtrate (applicazione di caratteri di escape) prima della loro iniezione nelle query.

Per quanto riguarda le SQL Injection di tipo 2, ossia quelle che utilizzano dati già presenti in database per generare altre query, deve essere adottata una accortezza di sviluppo: tutte le query che vengono eseguite sul database fanno riferimento a campi chiave interi e non vengono mai riutilizzati campi testuali da query precedenti per generarne altre. In questo modo si evita di avere problemi relativi a codice pericoloso, salvato nel database, permettendo comunque all'utente di avere la piena disponibilità di tutti i caratteri, anche quelli considerati pericolosi.

Inoltre, per evitare di rendere pubblica la struttura delle tabelle, attraverso l'interpretazione dei messaggi di errore, si ritiene opportuno disabilitare, sul server di produzione, qualsiasi messaggio di errore.

SR92. Il sistema deve nascondere qualsiasi messaggio di errore agli utenti finali (es. utilizzando la funzione *error_reporting(0)* di PHP).

2.2.9.3 Gestione della concorrenza e requisiti real-time

Data la natura condivisa di una applicazione web e quindi l'accesso concorrente alle medesime risorse, si rende necessaria l'adozione di tecniche per gestire la mutua esclusione delle richieste così da mantenere in ogni momento la consistenza e la coerenza delle informazioni presenti sul server. Il sistema permette a più utenti di visualizzare e gestire le informazioni relative ai dispositivi embedded e tale funzionalità si scontra necessariamente con la necessità di evitare che più utenti contemporaneamente modifichino le medesime informazioni.

SR93. Il sistema deve gestire transazioni concorrenti da parte di più utenti e segnalare se queste sono andate a buon fine o meno.

SR94. Il sistema deve garantire un tempo medio di risposta pari a 5 secondi in condizioni nominali.

SR95. Il sistema deve garantire, nella situazione di picco degli accessi, un ritardo nella risposta pari al massimo al 50% del tempo di risposta medio.

2.2.10 Tolleranza ai guasti

Per arrecare il minor disturbo all'utente finale e garantire la massima accessibilità del servizio, il sistema deve soddisfare i seguenti requisiti:

SR96. Il sistema deve garantire un up-time pari al 99.9% sia per quanto riguarda la disponibilità del server sia per la connettività.

Al fine di garantire che i dati presenti sui server saranno mantenuti consistenti e in sicurezza si rende necessaria l'adozione di tecniche di ridondanza delle informazioni (ad esempio hard-disk in configurazione RAID o server in configurazione cluster) per

ripristinare danni di lievi entità e la schedulazione di un processo di backup periodico per poter ripristinare il sistema in caso di guasti gravi.

SR97. Il sistema deve garantire il mantenimento (la consistenza) dei dati.

SR98. Gli errori dovuti a situazioni non previsti (errori di programmazione) non devono essere mostrati all'utente, ma devono essere riportati in un file di log apposito.

2.2.10.1 Manutenzione ed error recovery

Il sistema viene prevalentemente utilizzato in orario di ufficio (8 - 20) e quindi si ha a disposizione un'ampia fascia oraria per le operazioni di manutenzione ordinarie e straordinarie.

SR99. Il sistema deve poter essere aggiornato (manutenzione adattiva, perfettiva e correttiva) dalle 20 alle 8 in qualsiasi giorno dell'anno.

SR100. Il sistema deve mostrare una pagina di cortesia in cui si informa l'utente che il sistema è sottoposto a manutenzione e non è quindi accessibile.

3. Progettazione del sistema

Nei capitoli che seguono vengono descritte le interazioni tra i vari attori ed il sistema, attraverso i possibili casi d'uso, riferendoci alle specifiche dei requisiti utente. Per ciascuno di essi si ricorre ai seguenti diagrammi UML:

- **Use case diagram**: descrivono le interazioni possibili degli utenti col sistema, favorendo la comunicazione fra gli analisti e il cliente;
- **Class-view diagram**: descrivono le classi richieste per la realizzazione della funzionalità desiderata.

Per quanto riguarda i casi d'uso, questi sono descritti riportando la descrizione dell'operazione effettuabile dall'utente, seguita dal diagramma UML del caso d'uso e, infine, sono riportati i requisiti che vengono implementati.

3.1 Tipologie di attori

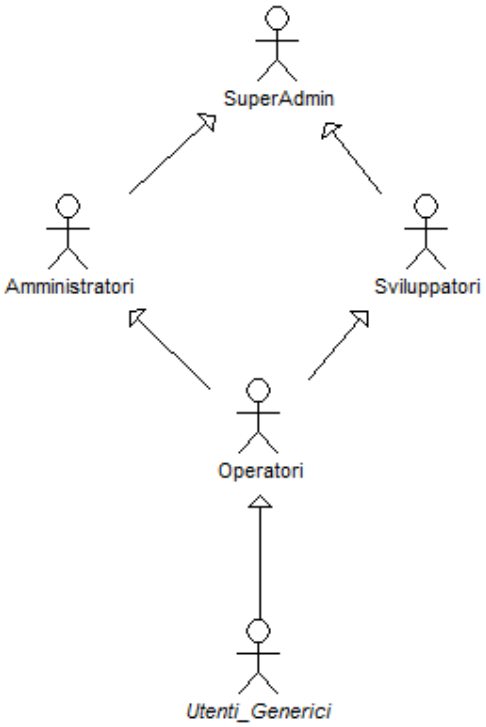


Figura 10 - Attori del sistema

Gli attori che interagiscono col sistema sono stati brevemente introdotti nel capitolo 2.1.1: come si nota dalla Figura 10, questi sono stati organizzati in modo da evidenziare le relazioni presenti. L'architettura esposta è di tipo incrementale, ovvero un utente generico è in grado di realizzare determinate operazioni, così come tutte le figure al di sopra di questo. Più nello specifico un utente di un certo livello può eseguire sia le operazioni del proprio livello sia le operazioni disponibili per gli utenti a livello più basso. Da questo enunciato si evince che il Super Amministratore è in grado di effettuare qualsiasi operazione sul sistema.

3.1.1 Utente generico

Ogni utente del sistema esegue alcune operazioni basilari, comuni a tutti gli utenti: al fine di descrivere queste interazioni in modo sufficientemente generico è stato scelto di considerare un utente astratto a cui assegnare queste funzionalità di base.

3.1.1.1 Sessione utente

L'utente generico deve essere in grado di autenticarsi al sistema, iniziando così la sua sessione di lavoro. Una volta deciso di concludere la sessione, può disconnettersi effettuando l'operazione di logout. Il caso d'uso associato è riportato in Figura 11.

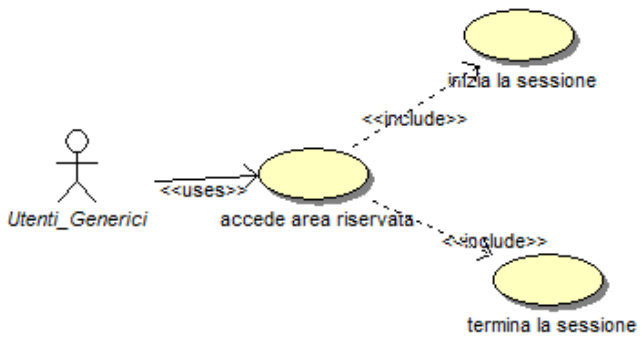


Figura 11 - Caso d'uso: sessione utente

I requisiti che questo caso d'uso implementa sono:

- *SR13 Un utente, utilizzando username e password, accede al sistema, qualora il suo account non risulti disabilitato. [UR2, UR4]*

3.1.1.2 Profilo personale

L'utente generico, una volta autenticato, accede al'area riservata e da qui può visualizzare il proprio profilo e modificare le informazioni collegate all'account personale. In Figura 12 è riportato il caso d'uso associato.

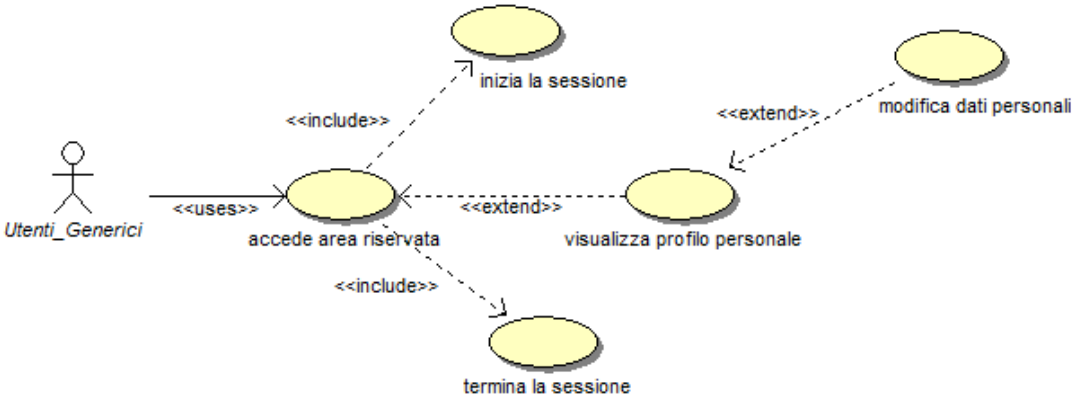


Figura 12 - Caso d'uso: profilo personale

I requisiti che tale caso d'uso implementa sono:

- *SR17 Deve essere possibile visualizzare tutte le informazioni memorizzate di un utente selezionato. [UR8]*
- *SR30 L'utente che ha avuto accesso al sistema deve poter modificare la propria password.*

3.1.2 Operatore

Il soggetto che usufruisce di limitate funzionalità offerte dal sistema è l'Operatore: l'accesso a tali funzionalità viene stabilito e concesso dall'Amministratore mediante il proprio pannello di back-end.

3.1.2.1 Gestione dispositivi

L'Operatore è il soggetto che si occupa, nel pratico, di gestire i dispositivi che gli vengono assegnati. In particolare egli ha accesso all'area di gestione dei dispositivi e può elencare tutti quelli su cui possiede il privilegio di accesso. Il caso d'uso in Figura 13 mostra le possibili interazioni tra l'Operatore e l'area della gestione dei dispositivi, indicando le singole operazioni effettuabili.

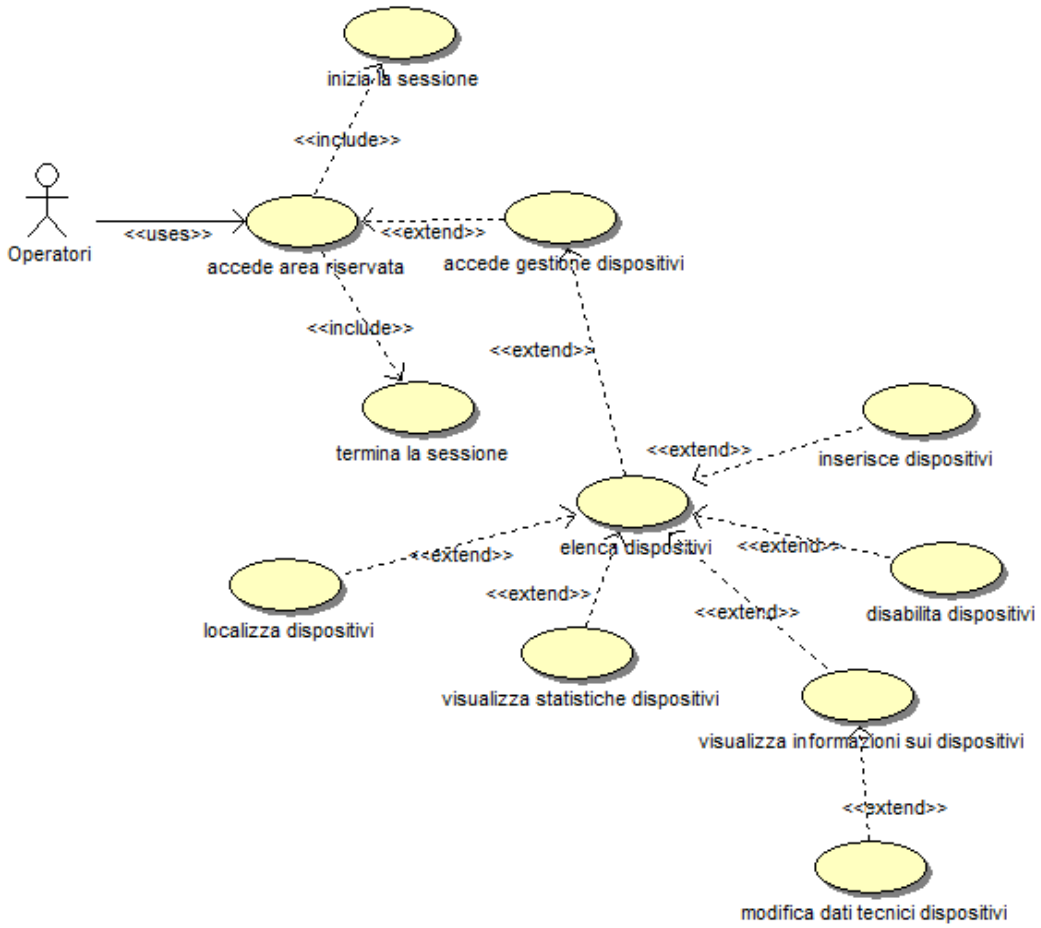


Figura 13 - Caso d'uso: gestione dei dispositivi

I requisiti che tale caso d'uso implementa sono:

- *SR43 L'utente SA, l'utente AM o un utente OP con privilegio, devono poter visualizzare la lista completa di tutti i dispositivi registrati nel sistema. [UR22]*
- *SR44 L'utente SA, l'utente AM o un utente OP con privilegio, devono poter inserire un nuovo dispositivo, specificando le informazioni che lo descrivono, in base al tipo di dispositivo. [UR23]*
- *SR45 L'utente SA, l'utente AM o un utente OP con privilegio, devono poter disabilitare un dispositivo esistente. [UR24]*
- *SR46 L'utente SA, l'utente AM o un utente OP con privilegio, devono poter abilitare un dispositivo disabilitato.*

- *SR47 L'utente SA, l'utente AM o un utente OP con privilegio, devono poter modificare i dati di un dispositivo in qualsiasi momento. [UR25]*
- *SR48 L'utente che può visualizzare la lista dei dispositivi di sistema, deve poter visualizzare le caratteristiche di un dispositivo scelto. [UR26]*
- *SR49 Specificando l'identificativo di un dispositivo, il sistema deve restituire la sua localizzazione entro il contesto organizzativo. [UR27]*
- *SR50 L'inserimento di un nuovo dispositivo è un'operazione che deve essere registrata nel log. [UR16]*
- *SR54 I campi sempre presenti che descrivono un dispositivo sono: identificativo univoco;posizione entro la struttura organizzativa.*

3.1.2.2 Operazioni con privilegio

GestDevice si configura come un sistema altamente personalizzabile con nuove funzionalità ed adattabile a qualsiasi esigenza: in virtù di questa importante caratteristica sono definiti dei privilegi personalizzati, ognuno collegato ad una operazione personalizzata.

Qualora un utente voglia accedere ad una operazione personalizzata deve possedere il privilegio richiesto, altrimenti il sistema restituisce un messaggio di errore in cui si segnalano le ragioni per cui l'accesso è ristretto.

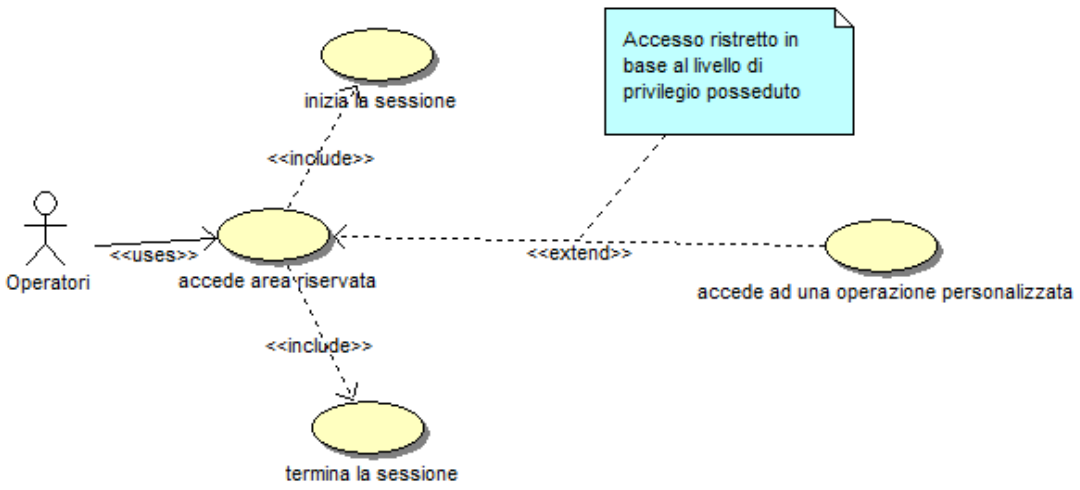


Figura 14 - Caso d'uso: operazioni con privilegio

Non sono presenti requisiti collegati a questo caso d'uso, in quanto tale funzionalità è frutto di una necessità di sviluppo.

3.1.1 Sviluppatore

La figura dello Sviluppatore si occupa di realizzare lo strato di software necessario a personalizzare il comportamento della piattaforma sulla base delle esigenze specifiche del contesto organizzativo in cui il sistema si troverà ad operare.

Lo Sviluppatore possiede l'accesso al sistema a livello di file e database e quindi, in linea teorica, può realizzare qualsiasi operazione.

3.1.1.1 Attività programmate

Un sistema altamente personalizzabile necessita di svolgere operazioni con cadenza costante o in un determinato momento. Il modulo di gestione delle attività programmate fornisce allo Sviluppatore un meccanismo per programmare tali compiti tramite una pratica interfaccia utente. In Figura 15 sono mostrate le operazioni disponibili per amministrare le attività programmate.

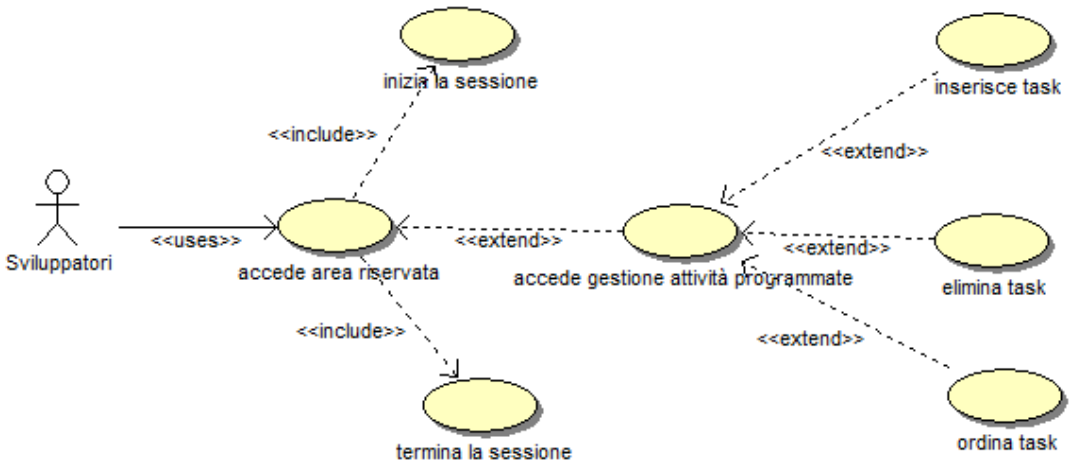


Figura 15 - Caso d'uso: attività programmate

I requisiti che tale caso d'uso implementa sono:

- *SR59 L'accesso al MGAP deve essere ristretto all'utente SA. [UR34]*
- *SR67 Deve essere possibile inserire un nuovo task, indicandone: [UR35]*

- *SR68 Deve essere possibile modificare i dati relativi ad un task, compreso l'ordine di esecuzione. [UR35, UR37]*
- *SR69 Deve essere possibile rimuovere un task. [UR36]*
- *SR70 Deve essere possibile disabilitare un task.*
- *SR71 Deve essere possibile abilitare un task.*

3.1.1.2 Personalizzazione comandi

Per semplificare l'aggiunta di nuove funzionalità a GestDevice, si prevede un'interfaccia utente che permette l'organizzazione delle operazioni disponibili in gruppi di comandi. Una volta acceduto un gruppo di comando sono elencate tutti i comandi disponibili ed è possibile aggiungerne di nuovi. Il menu dei comandi deve aggiornarsi automaticamente. In Figura 16 è riportato il caso d'uso che illustra le funzioni disponibili per l'utente che possiede il relativo privilegio.

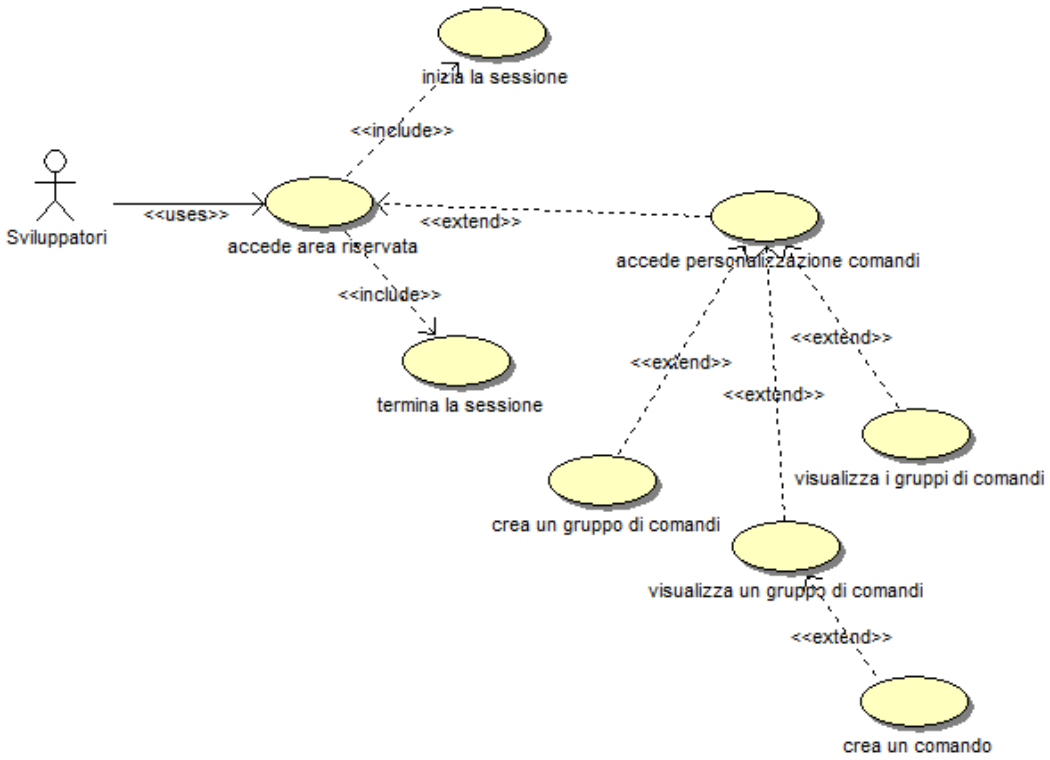


Figura 16 - Caso d'uso: personalizzazione comandi

Non sono presenti requisiti collegati a questo caso d'uso, in quanto tale funzionalità è frutto di una necessità di sviluppo.

3.1.1.3 Interpreti

Come descritto nel capitolo 2.2.5, GestDevice è in grado di comprendere qualsiasi formato di log, purché il sistema disponga di un opportuno interprete (parser). Lo Sviluppatore è il soggetto designato per la stesura del codice che implementa un interprete personalizzato. In appendice D.6.2 è possibile prendere visione del flusso di lavoro necessario alla creazione di un nuovo parser. In Figura 17 è riportato il caso d'uso associato.

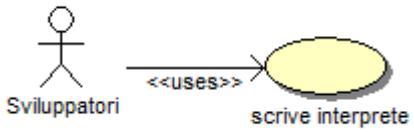


Figura 17 - Caso d'uso: scrittura di un interprete

I requisiti che tale caso d'uso implementa sono:

- *SR63 Il sistema deve permettere allo SV di realizzare ed installare un nuovo interprete sulla piattaforma.*
- *SR64 Il sistema deve permettere allo SV di modificare un interprete esistente.*
- *SR65 Il sistema deve permettere allo SV di rimuovere un interprete esistente.*
- *SR66 Il sistema deve mettere a disposizione dello SV di un interprete, una struttura dati in cui poter memorizzare informazioni di stato tra una esecuzione e l'altra dello stesso interprete. [UR29, SR3, SR4]*

3.1.2 Amministratore

L'Amministratore è un soggetto, designato dal Super Amministratore o da un altro Amministratore, che può accedere a tutte le funzionalità offerte dal sistema. Questo ruolo è svolto da un soggetto interno al contesto organizzativo entro il quale il sistema si colloca.

3.1.2.1 Amministrazione utenti

La gestione degli utenti è critica in un qualsiasi sistema informativo: l'Amministratore è il soggetto designato al coordinamento degli account utente e

deve poter, quindi, accedere a tutte le funzionalità utili ad una corretta gestione. In Figura 18 è riportato il caso d'uso illustrante le possibili operazioni effettuabili.

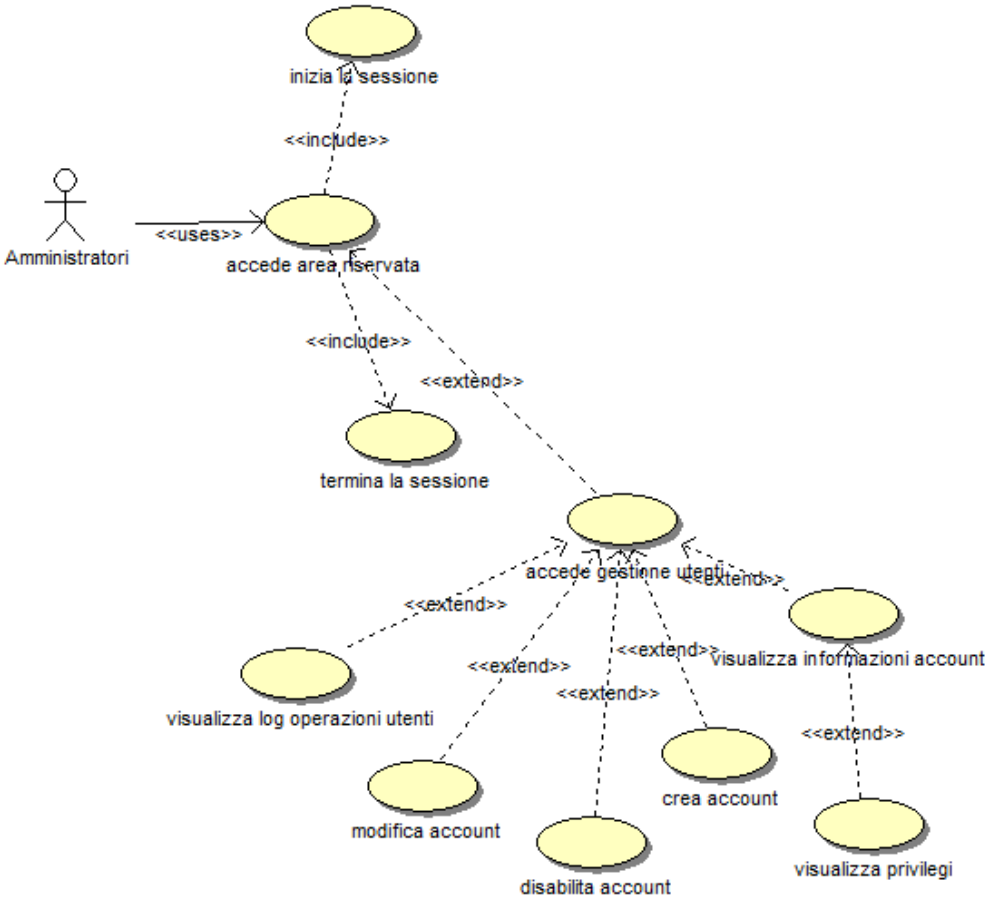


Figura 18 - Caso d'uso: amministrazione utenti

I requisiti che tale caso d'uso implementa sono:

- *SR5 L'accesso al MGU deve essere riservato agli utenti SA e AM. [UR1]*
- *SR6 I campi che descrivono un utente devono poter essere definiti da parte del SA.*
- *SR7 Le informazioni minime per descrivere un utente sono:*
- *SR8 Il MGU deve permettere la creazione di nuovi utenti specificando i valori per i campi che descrivono un utente, definiti con la funzionalità del requisito SR7. [UR3]*
- *SR9 Il MGU deve permettere la modifica di campi che descrivono un utente. [UR5]*

- *SR10 Il MGU deve permettere di disabilitare l'account di un utente, senza eliminarlo. [UR4, UR9]*
- *SR15 Il MGU deve prevedere una interfaccia per la visualizzazione di tutti gli utenti registrati nel sistema, compresi quelli disabilitati, evidenziandone la condizione di utente abilitato/disabilitato e il tipo di account (AM oppure OP). [UR7]*
- *SR19 La modifica delle informazioni di un utente esistente è un'operazione che deve essere registrata nel log. [UR16]*

3.1.2.2 Amministrazione gruppi

L'affiliazione degli utenti in gruppi semplifica la gestione di tutte le proprietà assegnabili agli stessi. Nello specifico l'amministrazione dei gruppi permette di associare un utente ad un gruppo, univocamente identificato, al fine di snellire il processo di notifica di messaggi o allarmi a più utenti contemporaneamente.

In Figura 19 è riportato il caso d'uso associato a questa funzionalità.

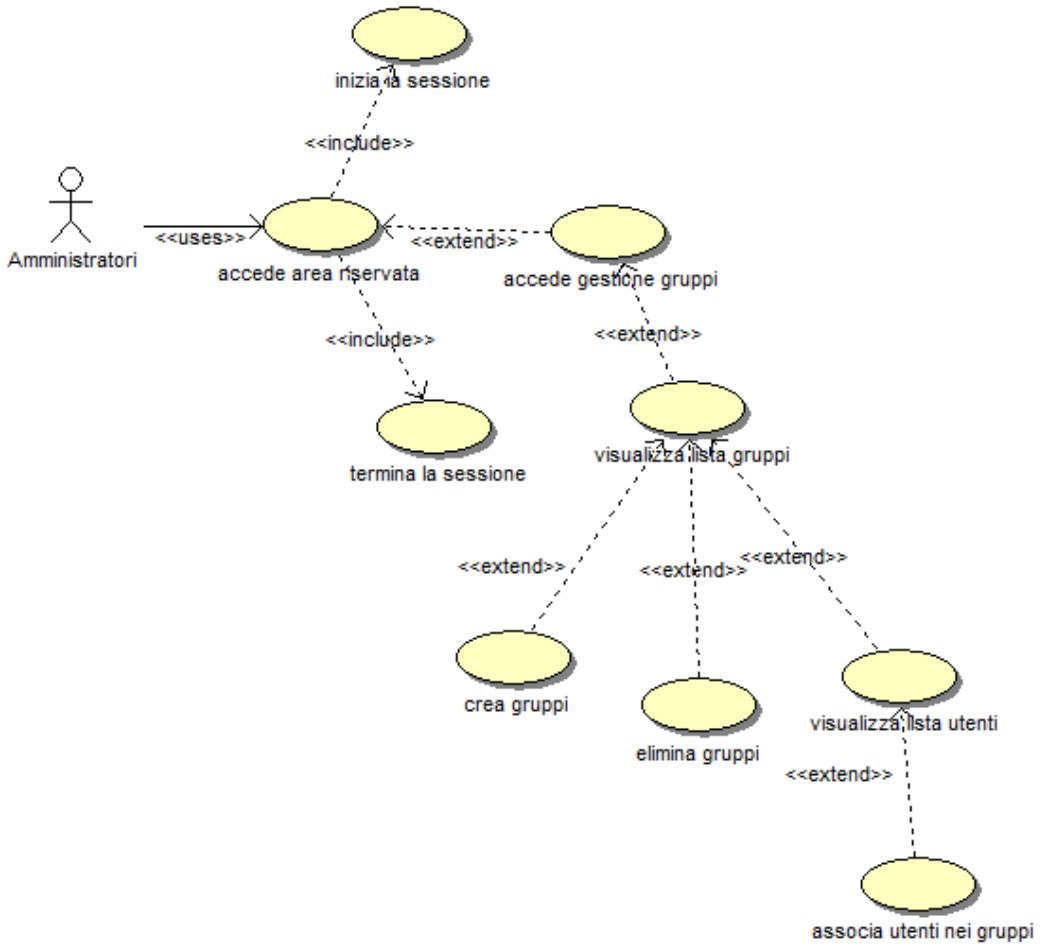


Figura 19 - Caso d'uso: amministrazione gruppi

I requisiti che tale caso d'uso implementa sono:

- *SR22 Il MGU deve permettere la creazione di un nuovo gruppo, indicandone il nome e gli eventuali utenti appartenenti. [UR10, UR11]*
- *SR23 Il MGU deve permettere di rimuovere un gruppo esistente, disassociando gli utenti collegati allo stesso. [UR12]*
- *SR24 Il MGU deve permettere di aggiungere utenti ad un gruppo. [UR13]*
- *SR25 Il MGU deve permettere di rimuovere utenti da un gruppo. [UR13]*
- *SR26 L'interfaccia deve permettere di visualizzare tutti i gruppi disponibili nel sistema, indicando gli utenti che vi appartengono. [UR14, UR15]*

3.1.2.3 Amministrazione privilegi

La creazione di nuove funzionalità si scontra l'esigenza di limitare l'accesso solo a determinati utenti: il modulo di amministrazione dei privilegi consente all'Amministratore di creare nuovi privilegi da utilizzare nel codice sorgente che implementa la nuova funzionalità.

In Figura 20 è riportato il caso d'uso che illustra le funzioni disponibili.

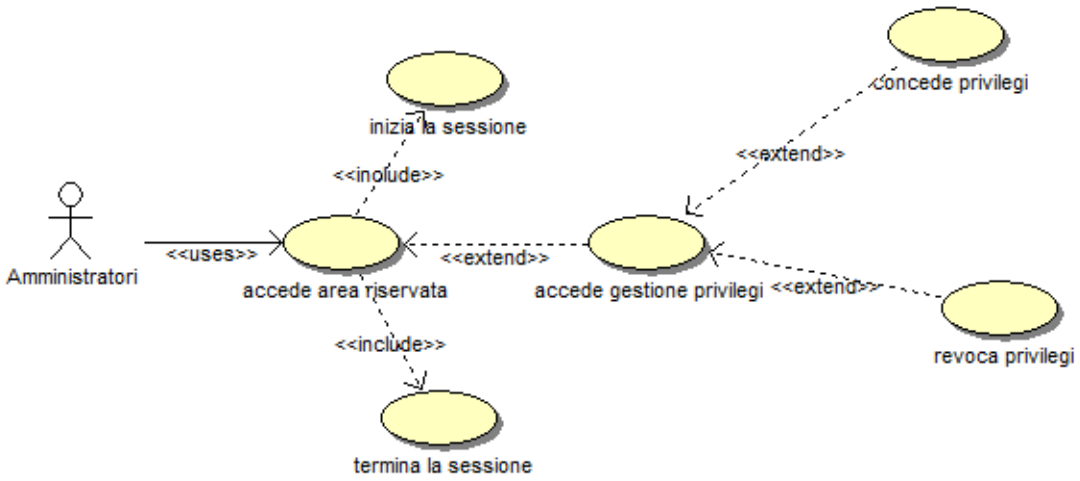


Figura 20 - Caso d'uso: amministrazione privilegi

I requisiti che tale caso d'uso implementa sono:

- *SR32 L'accesso al MGP deve essere riservato al SA e agli utenti AM. [UR18, UR19]*
- *SR33 Il MGP deve permettere la concessione di un privilegio esistente ad un utente. [UR18]*
- *SR34 Il MGP deve permettere di revocare un privilegio ad un utente. [UR19]*
- *SR35 Il MGP deve permettere di visualizzare l'elenco di tutti i privilegi personalizzati disponibili nel sistema. [UR21]*

3.1.3 Super Amministratore

Il Super Amministratore è colui che può accedere a tutte le funzionalità offerte dal sistema. Nello specifico è colui che supervisiona tutte le informazioni memorizzate e

gestite dal sistema e può intervenire eseguendo qualunque operazione messa a disposizione.

Solitamente questo ruolo viene svolto da un soggetto esterno al contesto organizzativo in cui il sistema si colloca. Il soggetto che ricopre questo compito si occupa quindi di supervisionare dall'esterno il corretto funzionamento del sistema, oltre alla consistenza delle informazioni, senza però interferire con lo svolgimento del flusso di lavoro gestito dal sistema.

3.1.3.1 Gestione unità organizzative

Per localizzare un dispositivo è necessario specificare la struttura dell'organizzazione entro la quale GestDevice si trova ad operare. Il Super Amministratore è l'unico soggetto in grado di modificare tale struttura.

In Figura 21 è illustrato il caso d'uso relativo.

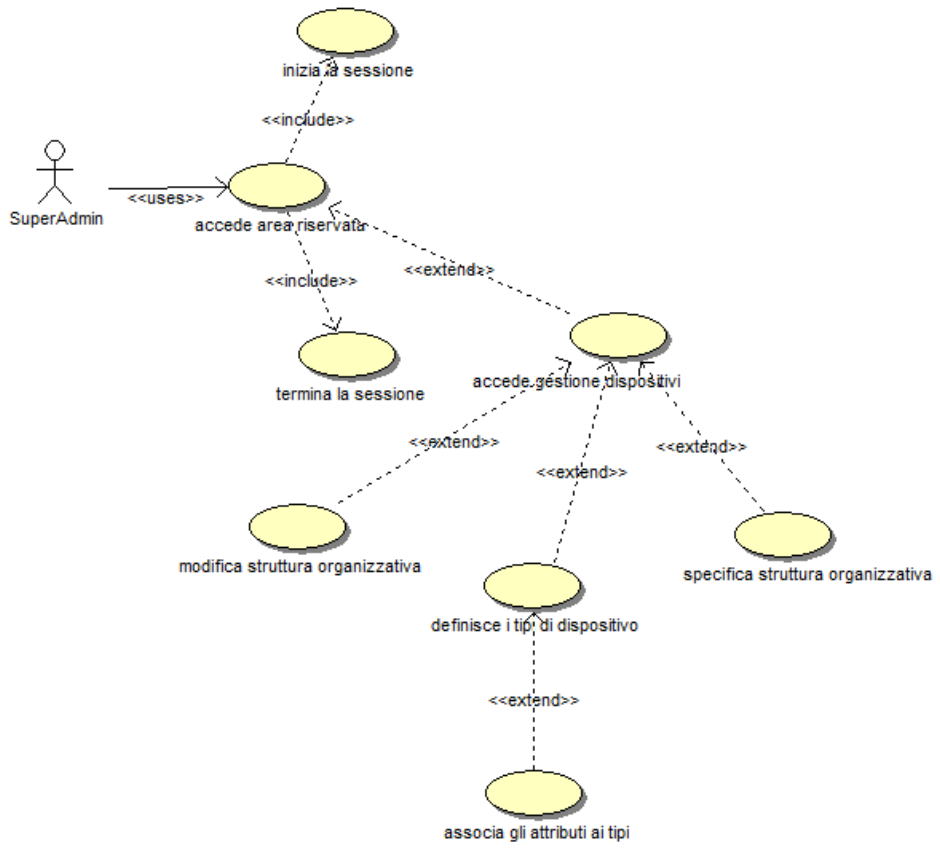


Figura 21 - Caso d'uso: gestione unità organizzative

I requisiti che tale caso d'uso implementa sono:

- *SR55 L'utente SA deve poter specificare la struttura organizzativa. [UR31, UR32]*
- *SR56 La modifica della struttura organizzativa può avvenire in qualsiasi momento, rispettando le politiche di consistenza dei dati.*
- *SR57 Un nodo della struttura organizzativa non deve poter essere modificato qualora esistano dispositivi associati allo stesso nodo o ad uno dei suoi discendenti.*
- *SR58 Un dispositivo deve poter essere localizzato anche su nodi non foglia della struttura organizzativa reale.*

3.2 Struttura dell'applicazione e mappe di navigazione

L'interfaccia grafica utente (GUI) consente all'utente di interagire con il sistema manipolando oggetti grafici convenzionali, quali aree di testo, bottoni e icone. Al giorno d'oggi un sistema che dispone di una interfaccia grafica non usabile o ne è completamente sprovvisto è destinato ad essere utilizzato da pochi. La progettazione del sistema parte proprio da questo elemento fondamentale per renderlo maggiormente user-friendly.

Nei capitoli che seguono viene analizzata l'architettura delle pagine in termini di contenuti e relazioni (links) fra le pagine stesse. L'architettura verrà studiata sia per quanto riguarda la struttura statica sia per quanto riguarda la parte inerente le interazioni fra gli utenti e il sistema. Questo ultimo aspetto viene trattato utilizzando le mappe di navigazione: queste trovano corrispondenza con quelli che sono i requisiti funzionali elencati in analisi.

3.2.1 Struttura statica

La struttura statica dell'applicativo è quella parte del sito che risulta sempre visibile all'utente durante la navigazione. In pratica è rappresentata da un template grafico che descrive la struttura di ogni pagina. Il modello è descritto tramite un diagramma UML che descrive una pagina generica composta di alcuni elementi come header,

menù di navigazione e contenuto. Ogni pagina deriva da questo modello aggiungendo le proprie funzionalità; come detto, queste sono descritte tramite mappe di navigazione.

Di seguito viene mostrato il diagramma delle classi UML della pagina generica.

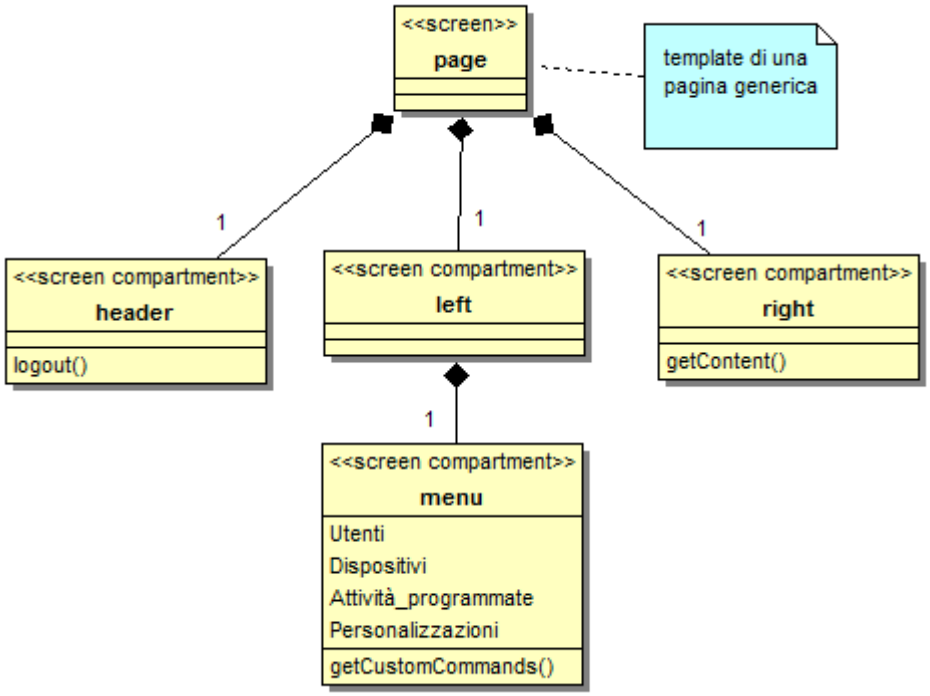


Figura 22 - Class diagram di una pagina generica

Un mock-up strutturale dell'applicativo è descritto nella figura che segue:

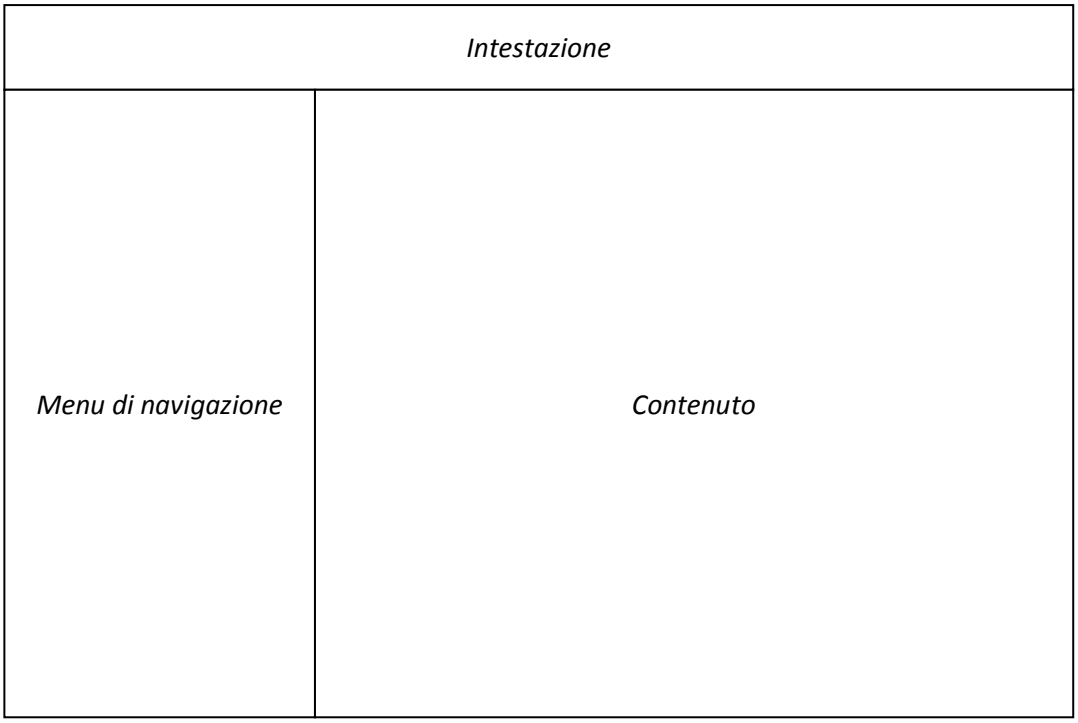


Figura 23 - Mock-up strutturale dell'interfaccia

La struttura di una pagina generica è stata creata per definire le parti di ogni pagina che saranno sempre presenti. I singoli elementi che sono presenti nella struttura sono organizzati secondo un modello a due colonne. Il menu di navigazione e l'intestazione sono sempre presenti, mentre il contenuto varia da pagina a pagina.

3.2.1.1 Intestazione

Si tratta di una barra che si estende in larghezza ed è posizionata nella parte alta dello schermo. All'interno di questo spazio vengono rappresentate le informazioni di login dell'utente e il comando di disconnessione.

- Titolo dell'applicativo;
- nome e cognome dell'utente autenticato;
- tipologia di account dell'utente autenticato;
- pulsante di disconnessione dal sistema.

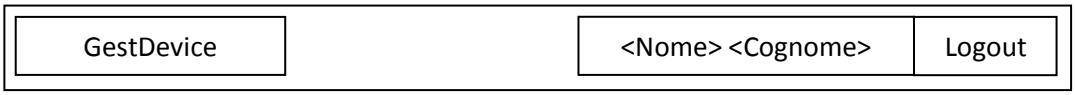


Figura 24 - Composizione dell'intestazione

3.2.1.2 Menu di navigazione

Il menu di navigazione è l'elemento che mostra all'utente le funzioni che può svolgere. Queste sono opportunamente filtrate in base ai privilegi di cui egli dispone. Le funzioni sono raggruppate in riquadri, utili per permette all'utente di individuare facilmente la funzione che sta cercando.

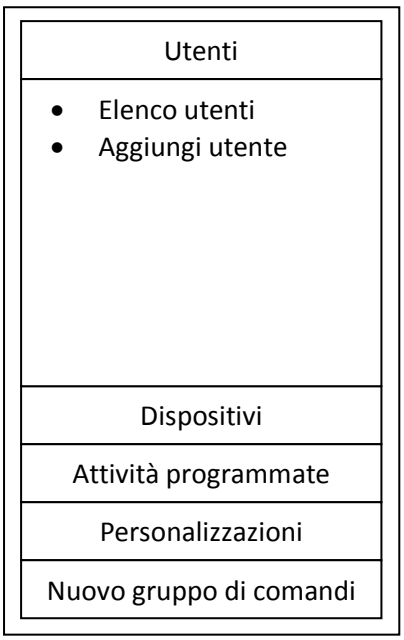


Figura 25 - Composizione del menu di navigazione

In Figura 25 si nota un riquadro chiuso, denominato "Nuovo gruppo di comandi": si tratta di un gruppo di comandi che racchiude operazioni realizzate ad hoc dallo Sviluppatore per implementare funzionalità richieste dall'utente.

3.2.1.3 Contenuto

Il layout prevede una parte centrale dedicata al contenuto in cui vengono visualizzate le informazioni inerenti alla singola pagina.

In generale una pagina mostra un titolo in alto al fine di rendere immediatamente chiaro all'utente il contenuto della pagina che ha richiesto. Subito sotto si trova un breve testo che illustra, più nello specifico, cosa è possibile trovare nella pagina e quali funzioni sono presenti.

3.2.2 Layout grafico

Una parte fondamentale per ottenere un sistema di qualità è sicuramente l'interfaccia grafica offerta agli utenti finali. Particolare cura deve essere presa per la progettazione del layout dato che rappresenta il biglietto da visita per il servizio di catalogazione e analisi di log dei dispositivi. I requisiti di usabilità spingono la progettazione verso dei criteri di successo come:

- Scelta del colore;
- Scelta di immagini e icone per i pulsanti di navigazione;
- Menu facili e intuitivi da navigare;
- Testi chiari e non veicolati dal solo colore e dalla forma (ad esempio usando tag come h1 per dare enfasi al testo);
- Informazioni sul funzionamento del servizio presenti in modo chiaro e inequivocabile;
- Risoluzioni visibili sulla maggior parte dei dispositivi con una profondità di colore adeguata.

Il sistema, rispettando i criteri sopracitati, deve risultare facile da apprendere e da ricordare, efficiente nel suo utilizzo e gradevole da usare.

3.2.2.1 Scelta del colore

La scelta del colore è sicuramente tra i requisiti di usabilità più importanti. Ogni colore trasmette un messaggio importante e quindi è fondamentale trasmettere all'utente finale la giusta sensazione anche in base a giusti accostamenti. Il cervello umano percepisce ed elabora i colori di un'interfaccia in meno di un secondo e registra una sensazione positiva o negativa che poi accompagnerà l'utente durante tutta la navigazione.

Nella ruota cromatica è possibile individuare degli accostamenti dei colori che trasmettono un significato universale ed inequivocabile. Ogni colore, quindi, non è interpretabile singolarmente ma viene indebolito o riceve forza da quelli circostanti.

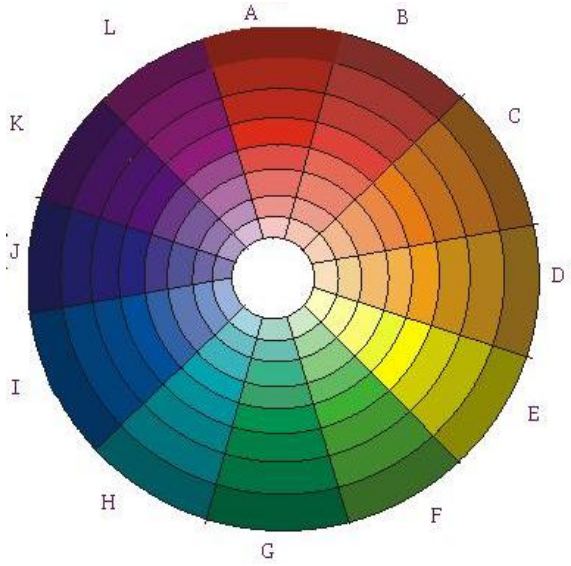


Figura 26 - Ruota cromatica

Un accostamento di colori utilizzabile è quello mostrato in Figura 27: si tratta di un modello monocromatico, composto da un giallo senape e un grigio sfumato.

Figura 27 - Accostamento di colori proposto

3.2.2.2 jQuery UI

jQuery UI fornisce astrazioni di basso livello di interazione e animazione, effetti avanzati altamente personalizzabili e widget. Si basa sulla libreria JavaScript jQuery e può essere utilizzato per creare applicazioni web interattive.

Il sito ufficiale (www.jqueryui.com) permette di comporre facilmente un tema grafico e di scaricare un file comprendente sia il core della libreria in formato js, sia i componenti grafici nell'aspetto desiderato.

Riprendendo l'esempio proposto in Figura 27 si riportano alcuni elementi grafici da poter utilizzare nell'interfaccia utente.

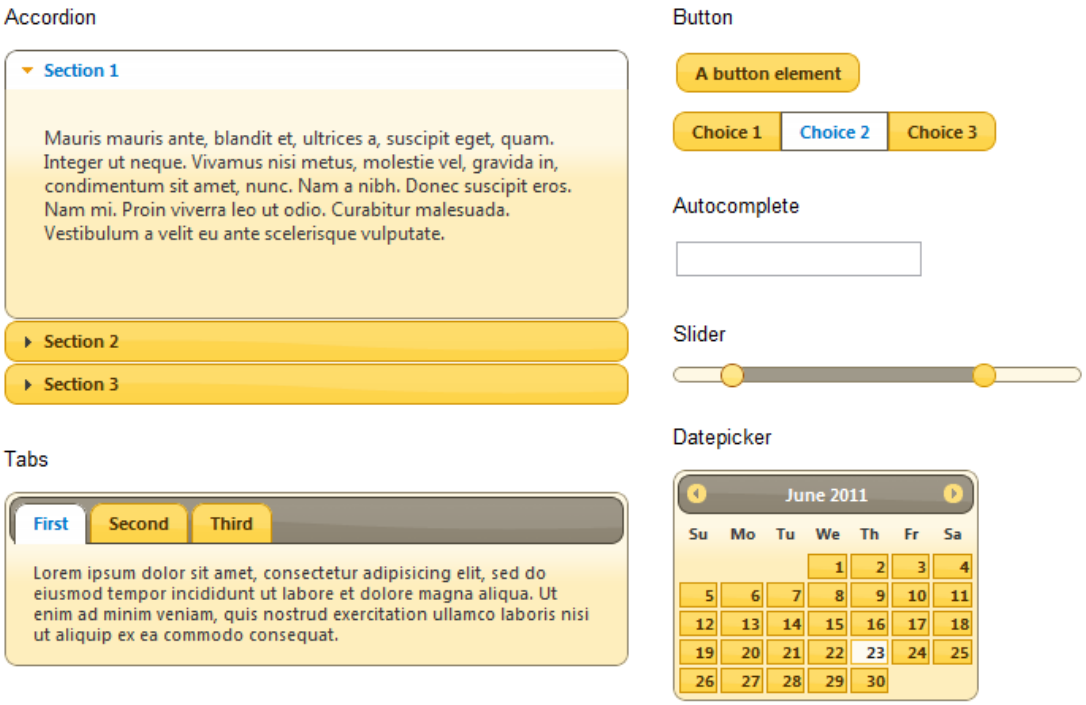


Figura 28 - Elementi grafici jQuery UI

Per modificare l'aspetto grafico di tutto il sistema è sufficiente scaricare dal sito di jQuery UI il pacchetto relativo al modello grafico prescelto e sostituire il file CSS e le immagini relative. È possibile anche far coesistere più modelli grafici di jQuery UI, semplicemente utilizzando un selettore CSS aggiuntivo.

3.2.3 Interazione tramite AJAX e JSON

Con l'avvento del Web 2.0 si è andata diffondendo una tecnica di sviluppo per la realizzazione di applicazioni web interattive (Rich Internet Application), molto più rapide nell'esecuzione rispetto al ricaricamento di tutta la pagina. Inoltre il programmatore non è vincolato a dover salvare tutte le informazioni di stato della pagina (es. riquadri aperti, selezioni effettuate in menu a discesa, etc.), ma può concentrarsi sullo sviluppo della logica di controllo dell'operazione in corso.

Lo sviluppo di applicazioni HTML con AJAX, sinonimo di Asynchronous JavaScript and XML, si basa su uno scambio di dati in background fra web browser e server come se si trattasse di una normale richiesta http effettuata in seguito alla pressione di un'ancora. AJAX è asincrono nel senso che i dati necessari

all'aggiornamento dell'interfaccia sono richiesti al server e caricati in background senza interferire con il comportamento della pagina esistente. Normalmente le funzioni richiamate sono scritte con il linguaggio JavaScript.

Il formato di risposta di una richiesta AJAX non è detto che sia puro HTML: quando si aggiorna l'interfaccia utente è raro dover modificare soltanto il contenuto di un contenitore (un div ad esempio), ma piuttosto si rende necessario il passaggio di molte informazioni. Ad esempio, un'informazione necessaria in ogni risposta AJAX è il fatto se l'utente è correttamente autenticato sul sistema. In caso negativo, il sistema dovrebbe mostrare una finestra di dialogo che permette all'utente di eseguire nuovamente l'autenticazione.

Il miglior modo per rispondere al browser con molte informazioni è l'utilizzo di JSON: la notazione degli oggetti JavaScript (JavaScript Object Notation) consente di passare una variabile esistente nel linguaggio server-side al linguaggio client-side (JavaScript) che ha eseguito la richiesta AJAX.

In Javascript è possibile effettuare richieste AJAX utilizzando una sintassi jQuery del tipo:

```
$.ajax({
    url: 'http://gestdevice.it/get_content.php',
    success: function( data ) {
        alert("Messaggio ricevuto!");
    }
});
```

Un esempio di pagina in linguaggio server-side che consente di rispondere alla richiesta mostrata di seguito:

```
<?php
    $response = array(
        'user_logged' => false,
        'date' => "11/05/2011",
        'html' => "<div id=\"my_div\">Nuovo contenuto da inserire
dove necessario.</div>";
    );

    echo json_encode($response);
?>
```

La risposta che il precedente script genera ha una forma simile a questa:

```
{"cmd":"list_users","user":{"logged":true,"data":[]},"html":""}
```

3.2.3.1 Interfaccia in JavaScript

Trattandosi di un'area di amministrazione, e come tale riservata solo ad una cerchia ristretta di utenti, è possibile utilizzare le potenzialità offerte da un'interfaccia completamente realizzata in JavaScript. Tale scelta implementativa si scontra necessariamente con la necessità per gli utenti di disporre di un browser con supporto JavaScript abilitato.

Qualche anno fa questo sarebbe stato uno dei principali motivi che avrebbero portato all'abbandono di questa idea di sviluppo: al giorno d'oggi JavaScript è utilizzato praticamente dal 100% dei browser in circolazione e questo non rappresenta quindi un problema.

Data	JavaScript Attivo	JavaScript Disattivo
2008	95%	5%
2007	94%	6%
2006	90%	10%
2005	89%	11%
2004	92%	8%
2003	89%	11%
2002	88%	12%
2001	81%	19%
2000	80%	20%

Tabella 3 - Statistiche di utilizzo di JavaScript

3.2.4 Mappe di navigazione

Le funzionalità offerte agli utenti saranno descritte tramite le mappe di navigazione, fornendo un utile supporto allo sviluppatore che implementerà le interfacce.

3.2.4.1 Accesso

L'interfaccia necessaria all'utente per autenticarsi al sistema è composta da una finestra di dialogo che include un form in cui poter inserire le credenziali di accesso:

- Nome utente;
- Password.

Qualora l'utente inserisca credenziali errate il sistema mostra nuovamente il form di immissione. L'inserimento di credenziali corrette permette l'accesso all'area riservata.

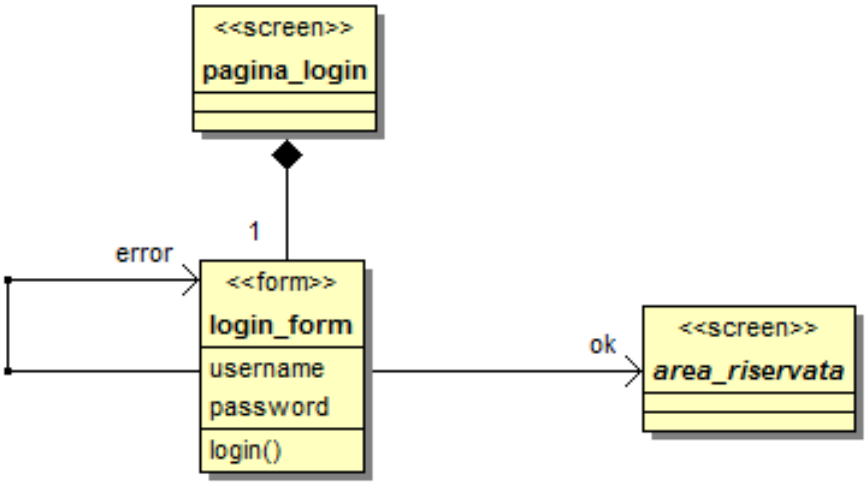


Figura 29 - Mappa di navigazione: accesso

3.2.4.2 Profilo personale

Ogni utente del sistema deve essere in grado di gestire il proprio profilo personale al fine di rendere aggiornate le informazioni. Un'operazione necessaria in un sistema di account è la modifica password: il form richiede all'utente l'inserimento della password attuale e consente l'inserimento della nuova password, con relativa conferma.

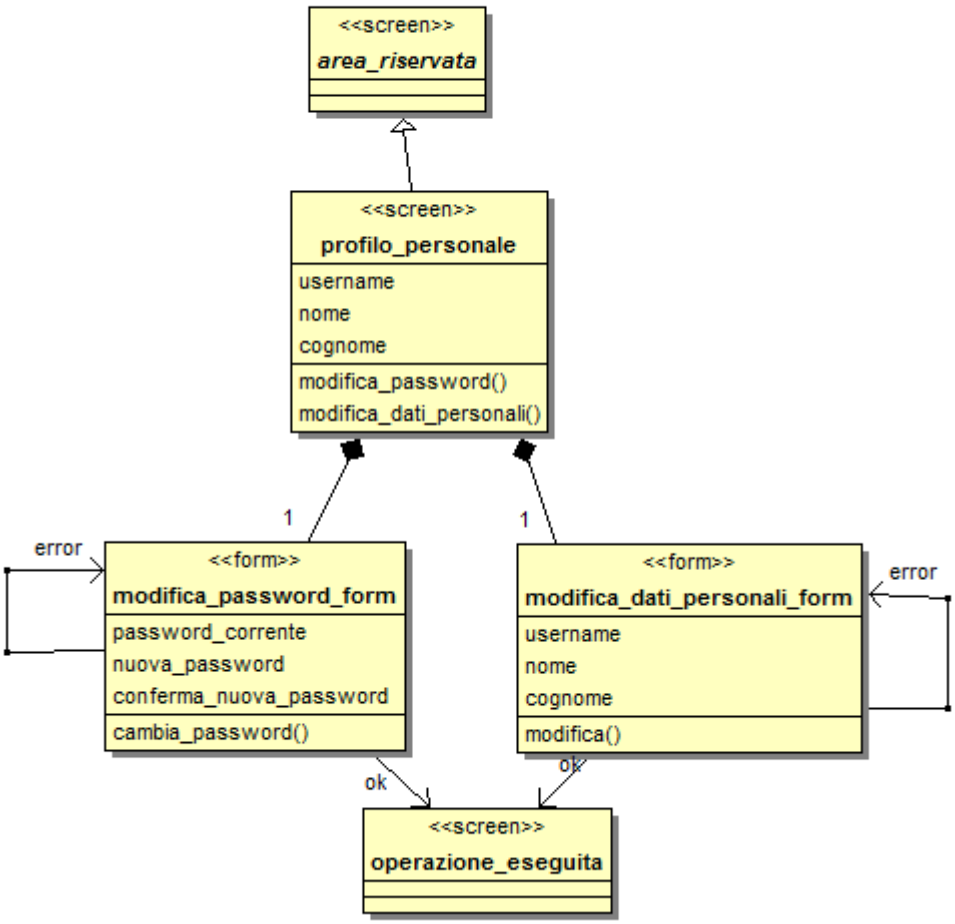


Figura 30 - Mappa di navigazione: profilo personale

3.2.4.3 Modulo gestione utenti

Il modulo consente di creare o modificare utenti utilizzando un form di inserimento. I dati degli utenti presenti sono riassunti in una pratica tabella, mostrando le caratteristiche peculiari di ognuno di essi: nome utente, privilegi, etc.

Inoltre è possibile amministrare i gruppi, crearne di nuovi e associare gli utenti negli stessi.

I privilegi creati (vedi capitolo 3.2.4.7) possono essere assegnati ad ogni utente tramite un semplice form.

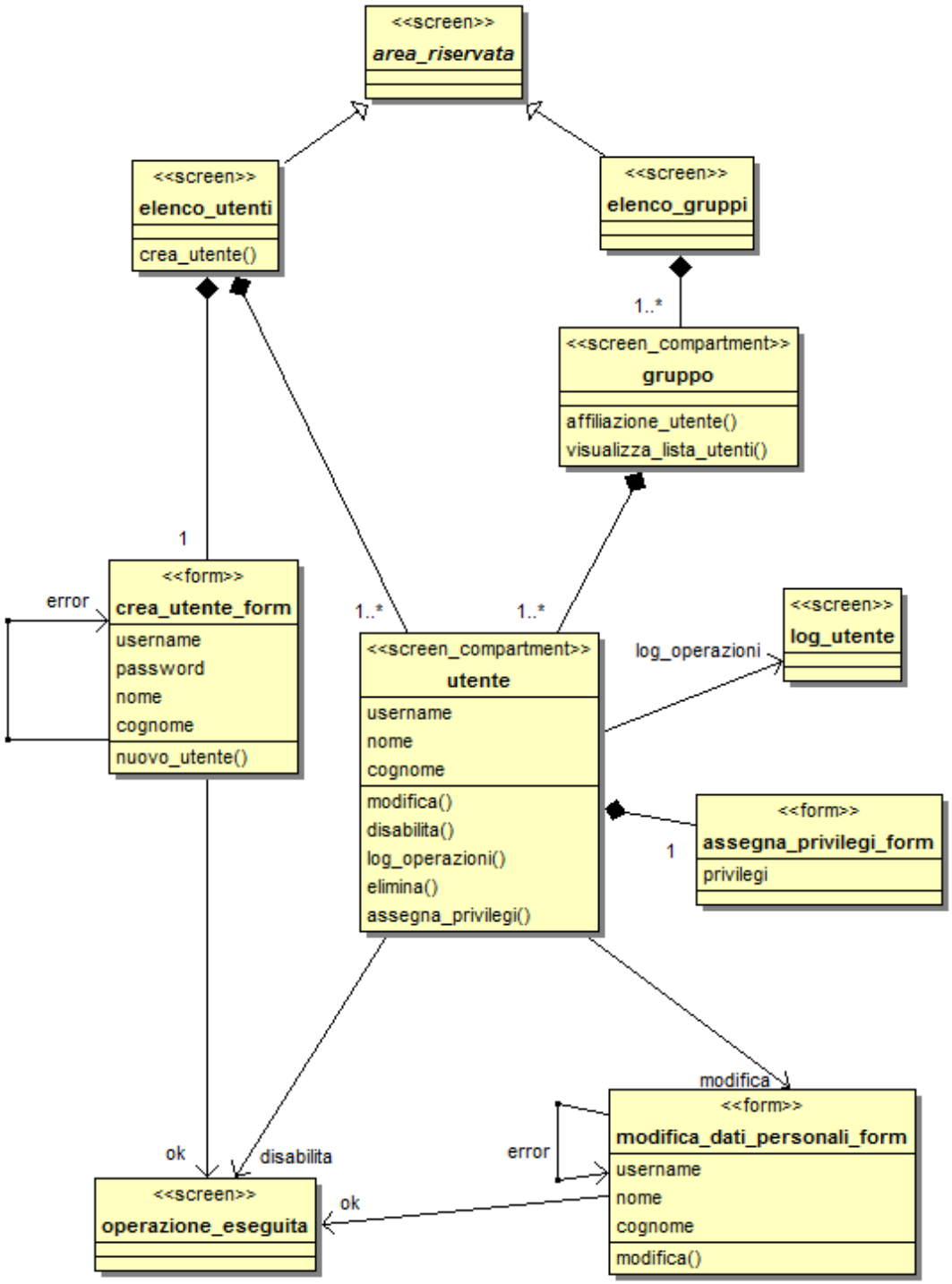


Figura 31 - Mappe di navigazione: gestione utenti

3.2.4.4 *Modulo gestione dispositivi*

L'interfaccia di amministrazione dei dispositivi permette di creare nuovi dispositivi, specificando i campi che li descrivono. Varie sono le operazioni che possono essere

eseguite su un dispositivo: localizzazione, disabilitazione o visualizzazione di statistiche.

L'operazione di localizzazione necessita della specifica di come è organizzata l'azienda che utilizza GestDevice: una pratica interfaccia permette di gestire i livelli organizzativi.

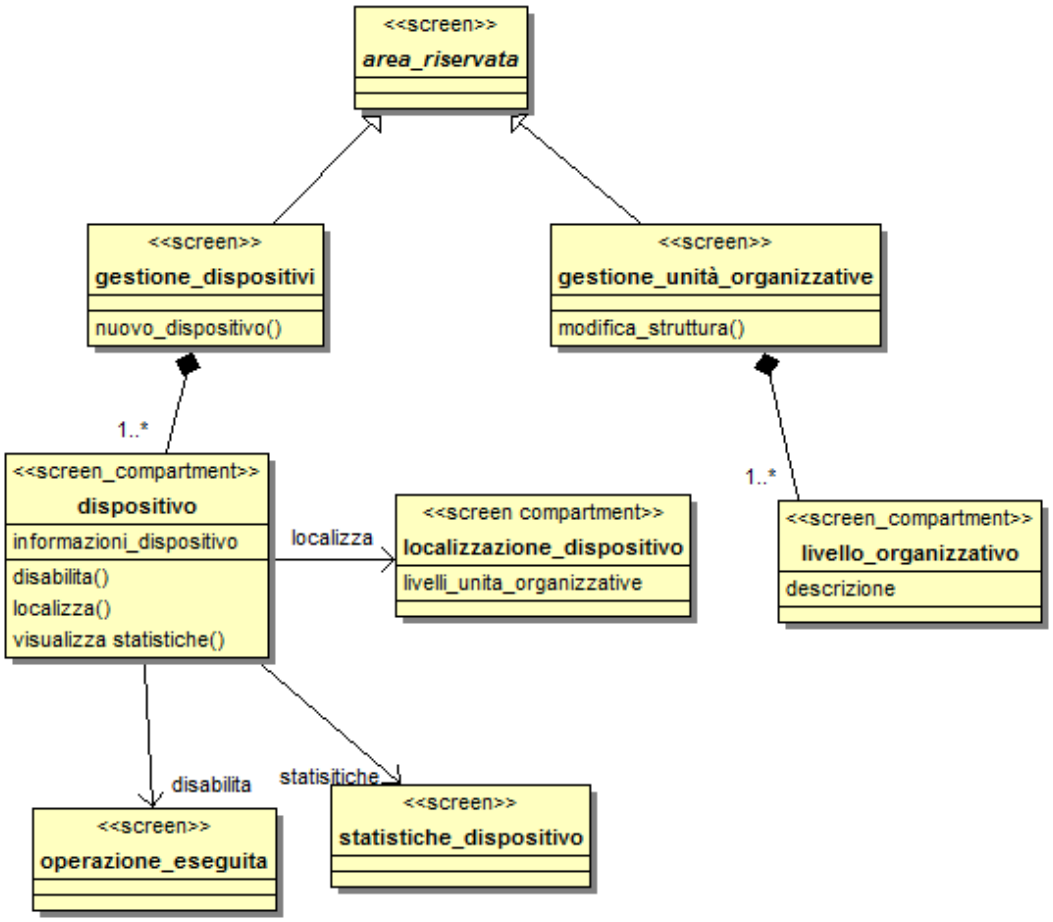


Figura 32 - Mappa di navigazione: gestione dispositivi

3.2.4.5 Modulo attività programmate

L'utilizzo di attività programmate consente allo sviluppatore di pianificare operazioni da eseguire in un determinato istante temporale.

L'interfaccia consente di elencare tutte le attività programmate già presenti, permettendo di inserirne di nuovi, oltre che modificare l'ordine di esecuzione.

I campi necessari a specificare una nuova attività sono:

- Descrizione;
- Script che implementa l'operazione da eseguire;
- Valori temporali per conoscere l'istante in cui deve essere eseguita l'operazione.

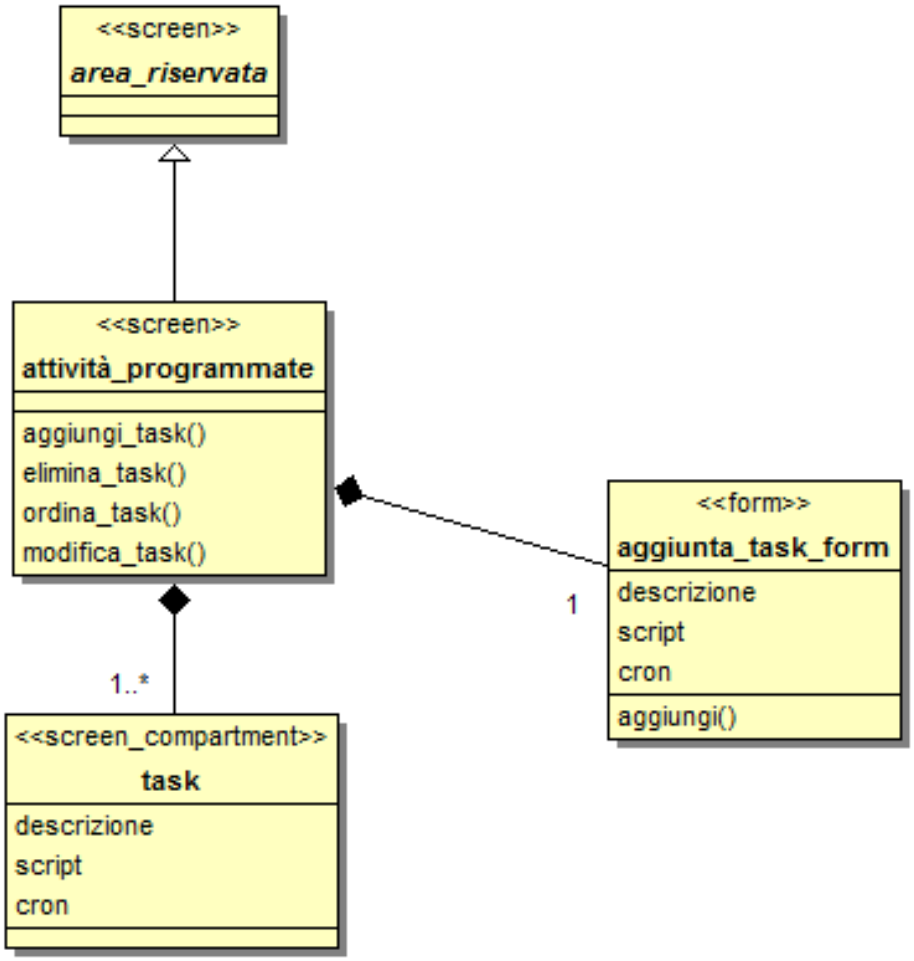


Figura 33 - Mappa di navigazione: attività programmate

3.2.4.6 *Personalizzazione comandi*

Lo Sviluppatore ha a disposizione un'interfaccia per estendere le funzionalità offerte con un metodo poco invasivo per la stabilità di GestDevice.

Un form di inserimento permette di creare nuovi comandi, specificandone il privilegio di accesso richiesto.

Un altro form consente di raggruppare i comandi esistenti in gruppi di comandi.

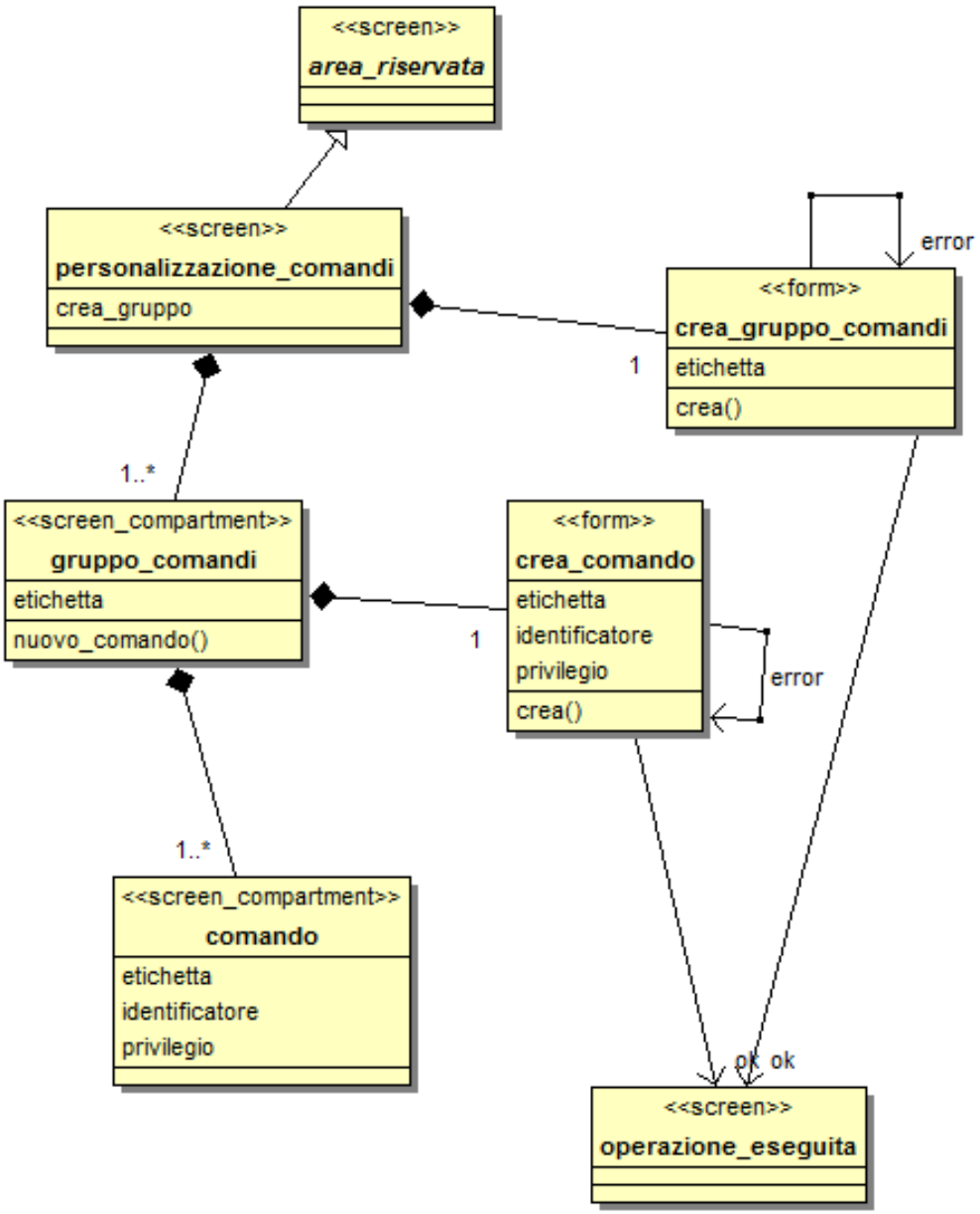


Figura 34 - Mappa di navigazione: personalizzazione comandi

3.2.4.7 Modulo gestione privilegi

La creazione di nuove funzionalità incontra il problema di restringerne l'accesso solo a determinate categorie di utenti: per automatizzare il processo, lo sviluppatore crea nuovi privilegi specificano un'etichetta a scopo mnemonico.

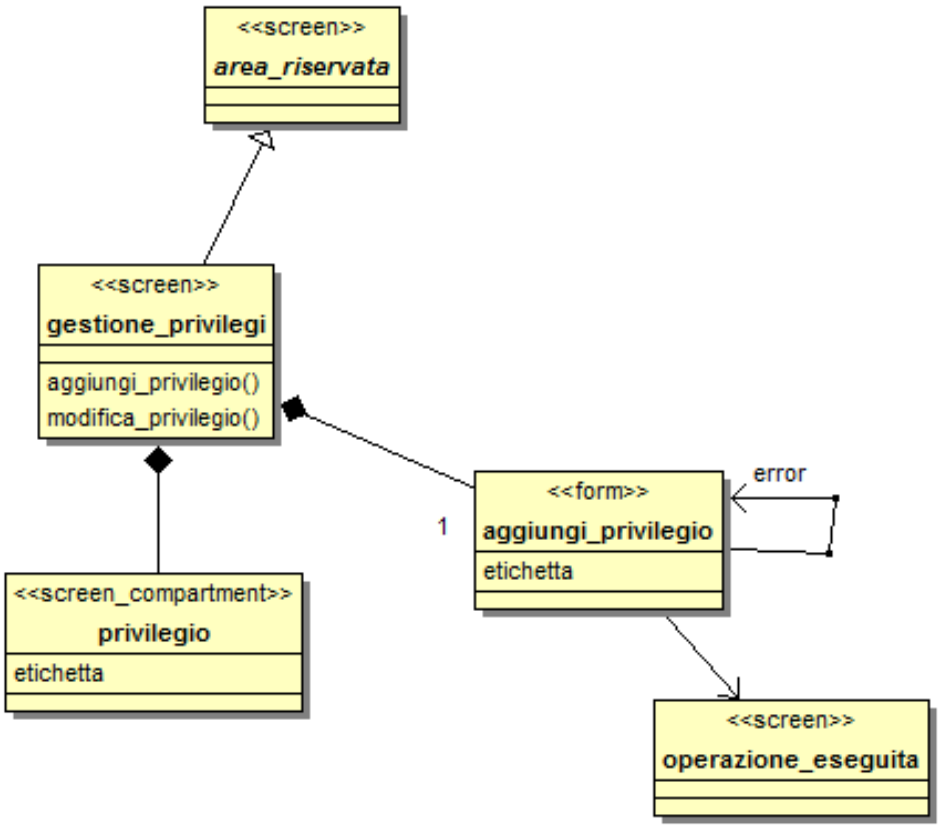


Figura 35 - Mappa di navigazione: privilegi

3.3 Progetto software

Nei capitoli che seguono vengono illustrate le classi che permettono di rappresentare oggetti con un insieme definito di proprietà, la struttura del database utilizzata, le funzionalità che il Core mette a disposizione dello Sviluppatore, la gestione del log degli eventi, oltre ad esempi per personalizzare alcune funzionalità del sistema.

3.3.1 Classi per la rappresentazione di oggetti

Il sistema deve rappresentare moltissimi oggetti differenti tra loro, ognuno con proprietà diverse:

- **Utente**: rappresentato da un nome utente, una password, le generalità anagrafiche, etc.;
- **Dispositivo generico**: rappresentato da un identificativo, un numero seriale, la marca, etc.;

- **Attività programmata**: rappresentato da una descrizione, uno script PHP e alcune regole di temporizzazione necessarie all'esecuzione.

Oltre agli oggetti mostrati nel precedente elenco potrebbe rendersi necessario rappresentare altri tipi di oggetti: per questo motivo è necessario realizzare un insieme di classi che permettano di creare e salvare oggetti in database di una qualsiasi tipologia.

In particolare le classi che implementano questo meccanismo sono:

- class.instance.php
- class.content.php
- class.content_attribute.php
- class.content_attribute_link.php
- class.content_attribute_value.php
- class.content_enum.php
- class.content_relation.php
- class.content_type.php

In Appendice A è possibile prendere visione del codice sorgente dei file sopra citati.

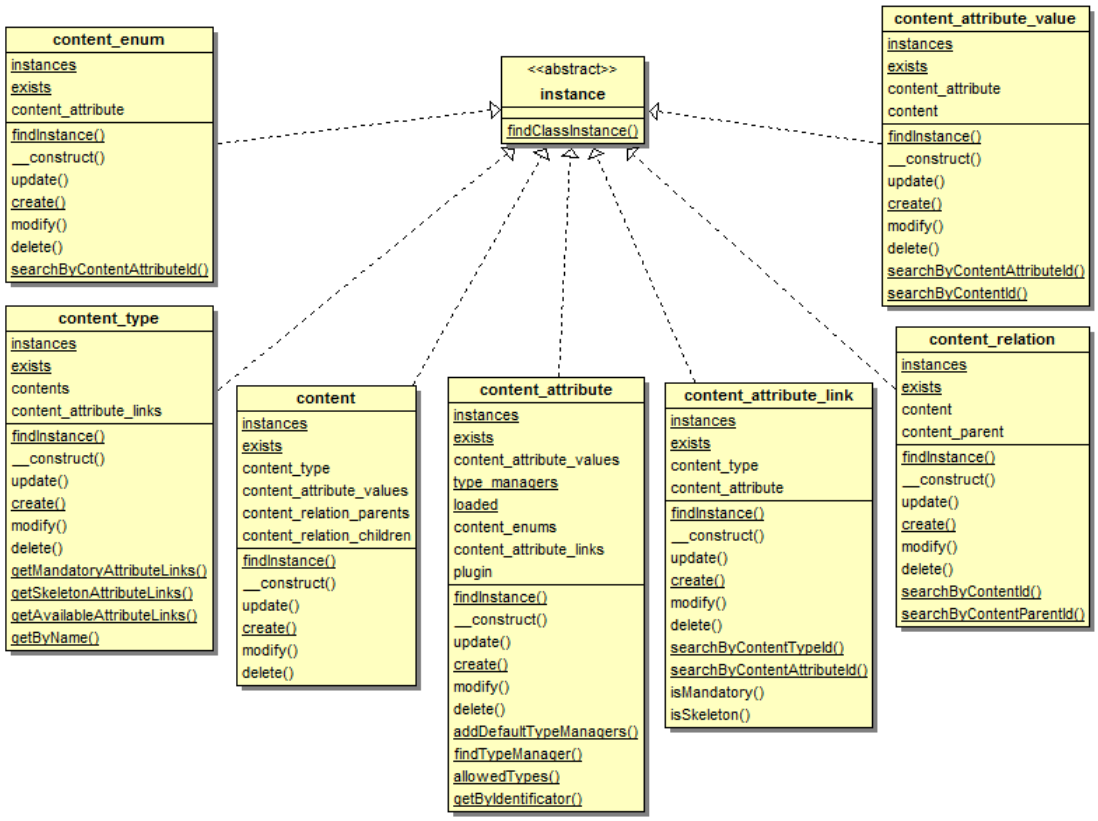


Figura 36 - Diagramma UML delle classi per la rappresentazioine di oggetti generici

Ogni oggetto classe con il prefisso “content_” rappresenta un oggetto della tabella omonima in database (vedi capitolo 3.3.2).

Gli oggetti si instanziano chiamando il metodo *findInstance()*, anziché invocare direttamente il costruttore. Questa precauzione è necessaria per utilizzare il meccanismo di caching implementato dalla classe astratta instance. Ogni classe è fornita di un attributo membro denominato instances, che memorizza le informazioni recuperate dalla relativa tabella in database al fine di evitare inutili query in database per ottenere informazioni su oggetti che già sono stati richiesti.

Ogni classe dispone, inoltre, dei metodi:

- *update()*: permette di recuperare le informazioni dalla entry nella tabella opportuna in database al fine mantenere aggiornato l’oggetto;
- *create()*: permette di generare una nuova entry nella tabella relativa, specificando i campi necessari;

- *modify()*: permette di modificare le proprietà di un oggetto e di salvarle in database;
- *delete()*: permette di eliminare l'oggetto corrente e di rimuovere la relativa entry in database.

In particolare, invocando il metodo *update()*, l'oggetto viene ad essere popolato da attributi membro denominati esattamente come i campi della relativa tabella. Ad esempio, la tabella contents contiene le informazioni relative agli oggetti che vogliamo rappresentare (vedi capitolo 3.3.2) e quindi, la generazione o l'aggiornamento di un oggetto renderà disponibili gli attributi membro *content_id*, *content_type_id*, *content_name* e *content_creation_date_time*.

Inoltre sono presenti alcuni attributi membro particolari, che utilizzano una particolarità del linguaggio di scripting utilizzato (vedi capitolo 2.2.7.2.1): si tratta di attributi che, qualora non esistano alla prima chiamata, vengono popolati attraverso una specifica funzione appositamente predisposta. Questo meccanismo permette di aggiornare molto rapidamente le informazioni che rappresentano un oggetto, semplicemente resettandone il valore. Il meccanismo che sta alla base di quanto esposto prende il nome di magic function, in particolare la *__get()*.

Di seguito è mostrato un esempio di utilizzo di tale magic function:

```
<?php
    class MyClass
    {
        // ...

        public function __get($name)
        {
            switch($name)
            {
                case "my_attribute":
                    return "a value for my_attribute"
                    break;
            }
            trigger_error("Undefined property '$name' in " .
get_class($this));
        }
    }
?>
```

Instanziando un oggetto di tipo MyClass e richiamando l'attributo membro "my_attribute" di ottiene una stringa che riporta "a value for my attribute".

```
<?php
    $c = new MyClass();
    $value = $c->my_attribute;
    // Value contains "a value for my attribute"
?>
```

Abbiamo detto che ogni contenuto è rappresentato dai valori che assumono i suoi attributi: ad esempio, per un utente, sono presenti i campi "username", "password", "indirizzo email", "data di creazione" e altri.

Ognuno di questi può avere un formato diverso dagli altri. Analizziamo i campi appena esposti per introdurre i gestori di tipi di attributo:

- **username**: si tratta di una stringa composta da caratteri specifici, solitamente compresi nell'intervallo [a-z0-9_] con varie regole di composizione come, ad esempio, il numero minimo di caratteri o il divieto di iniziare per "_";
- **password**: composta da un minimo numero di caratteri alfanumerici, può comprendere anche un salt che non viene mostrato all'utente;
- **indirizzo email**: stringa con una precisa regola sintattica indicante la casella di posta elettronica, seguita dal carattere "@" e dal dominio di appartenenza;
- **data di creazione**: campo composto che include il giorno, il mese e l'anno in cui l'utente è stato creato.

Riportiamo di seguito il codice sorgente del gestore astratto di tipi di attributo, da cui derivano i gestori personalizzati.

```
<?php

interface IAttributeTypeManager
{
    /**
     *  Used to define if a ContentAttributeType can be managed by
     *  the plugin which implements this class
     */
    public static function canHandle(ContentAttribute $attribute);
```

```
}

abstract class AttributeTypeManager implements IAttributeTypeManager
{
    // Reference to ContentAttribute
    var $content_attribute;

    public function __construct(ContentAttribute $ca)
    {
        $this->content_attribute = $ca;
    }

    /**
     *  It populates the element of the 'as-ed' element by ref
     *  through $destination_property
     */
    public abstract function parseValue(&$destination_property,
ContentAttributeValue $value);

    /**
     *  Creates specific properties inside the given object.
     *  (explode and parse the content_attribute_field_field)
     */
    public abstract function objectConstructor(ContentAttributeValue
$value);

    /**
     *  Used to validate the given value in the admin area.
     *  It accepts a string or an array. If it's an array, every key
     *  is the name of an inner-field in the attribute.
     *  It validates every single value and returns true if there
     *  are no errors.
     *  Return false or an array in which every key is the name of
     *  an inner-field in the attribute and the value is a constant
     *  name for the given error (to localize it).
     *  e.g. $value = array("name" => "Marco", "surname" => "");
     *  @return array("surname" => "not_specified_surname")
     */
    public abstract function validateValue($value, $mandatory =
false);

    /**
     *  Get the html template to handle the attribute type
     *  If $values are passed, include them in the correct input
     *  fields
     */
    public abstract function adminTemplate($values);


    /**
     *  Used to generate the insert statement for a new attribute
     *  value
     */
    public abstract function insertValue($value);

    /**
     * Get an array containing:
     * array(
     *      "free_text" => array(
```

```
     *           array(
     *               "options" => "text_type=single_row",
     *               "label" => "Testo su linea singola"
     *           ),
     *           array(
     *               "options" => "text_type=multiple_row",
     *               "label" => "Testo su linea multipla"
     *           )
     *       ),
     *       ...
     * );
     */
    public static abstract function allowedTypes(&$arr);
}
?>
```

La classe AttributeTypeManager implementa l'interfaccia IAttributeTypeManager: il metodo canHandle è basilare per determinare l'applicabilità del gestore di attributo.

In particolare, analizziamo i metodi presenti, che ogni gestore di attributo deve definire:

- *objectConstructor()*: si occupa di aggiungere attributi al contenuto;
- *parseValue()*: popola l'attributo "parsed_attributes" con i valori che rappresentano l'attributo;
- *validateValue()*: si occupa di verificare i dati da salvare prima di effettuare la memorizzazione del valore dell'attributo;
- *adminTemplate()*: restituisce il modello grafico necessario in fase di inserimento e modifica dell'attributo;
- *insertValue()*: formatta i valori dell'attributo prima del salvataggio;
- *allowedTypes()*: elenca le tipologie di attributi che il gestore di attributo è in grado di riconoscere.

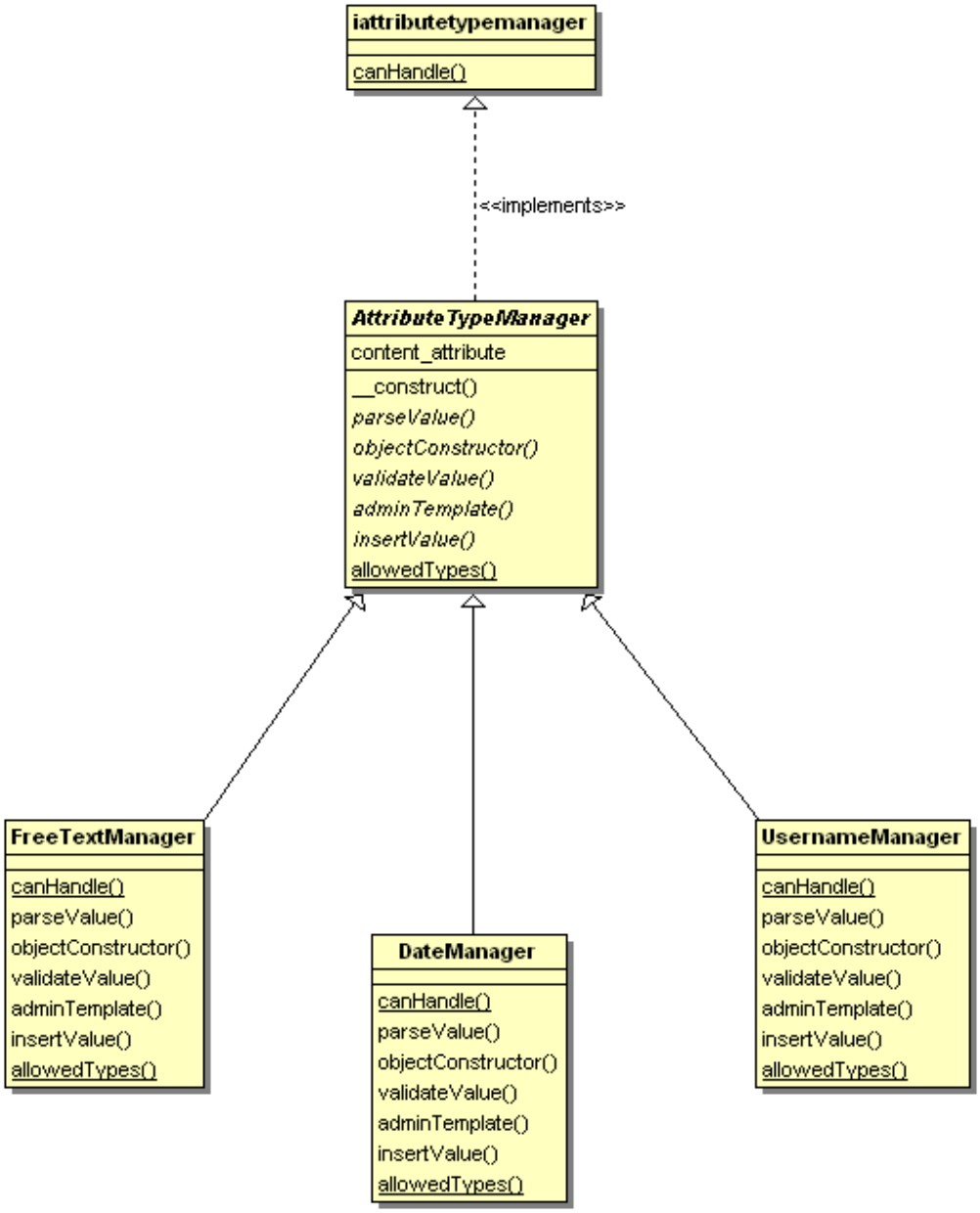


Figura 37 - Diagramma delle classi per i gestori di attributo

3.3.2 Progettazione del database

Una qualsiasi applicazione che fa uso di informazioni e dati necessari al suo funzionamento necessita di un contenitore sicuro per la loro memorizzazione. In particolare, una applicazione web, come GestDevice, utilizza un database per la conservazione e il reperimento delle informazioni necessarie al suo funzionamento: dati degli utenti, livelli di privilegio, assegnamento dei privilegi, etc. Utilizzando una

base di dati, non è necessario preoccuparsi di come tali informazioni vengono gestite.

Nei capitoli successivi verranno mostrate le fasi di progettazione della base di dati, il DBMS scelto e gli strumenti utilizzati.

3.3.2.1 Schema E-R

Tutte le caratteristiche esposte nei capitoli precedenti richiedono la memorizzazione di informazioni di stato al fine di descrivere, in modo consistente, gli oggetti e le relazioni tra gli stessi. Lo schema E-R che segue utilizza:

- tabelle con motore InnoDB;
- relazioni tra le stesse;
- vincoli di integrità referenziale.

In Figura 38 è riportato lo schema Entità-Relazioni utilizzato per GestDevice. Come si può notare esistono 3 aggregati, ognuno che implementa un gruppo di informazioni tra loro collegate:

- **contenuti**: tutte le tabelle col prefisso "contents_" sono necessarie per la descrizione di oggetti generici;
- **personalizzazione**: la tabella "commands", la tabella "commands_groups" e la tabella "privileges" sono necessarie per adattare il comportamento del sistema ad esigenze specifiche; nello specifico, sono utilizzate per descrivere comandi personalizzati e gestire le politiche di restrizione di accesso;
- **proprietà**: la tabella "properties" è designata a contenere le variabili persistenti (vedi capitolo 3.3.4.1).

Il primo aggregato, i **contenuti**, comprende le tabelle:

- contents: la tabella principale che racchiude i riferimenti agli oggetti che si sono creati;
- contents_types: necessaria per identificare le tipologie di oggetti che si possono descrivere;

- contents_attributes: gli attributi sono le entità necessarie a descrivere le caratteristiche di ogni tipologia di oggetto;
- contents_attributes_links: indica quali attributi descrivono i tipi; un campo specifico indica se l'attributo è obbligatorio oppure facoltativo;
- contents_attributes_values: contiene i valori degli attributi che descrivono gli oggetti;
- contents_enums: contiene i valori degli attributi a scelta multipla;
- contents_relations: la tabella permette di descrivere le relazioni che esistono tra gli oggetti; tramite questa tabella è possibile descrivere relazioni con più padri e/o più figli.

L'aggregato **personalizzazione** è formato dalle seguenti tabelle:

- privileges: contiene i privilegi da poter assegnare agli utenti e utilizzabili per limitare l'accesso alle funzioni personalizzate;
- commands_groups: indica i raggruppamenti dei comandi, al fine di organizzarli in base all'area di azione degli stessi;
- commands: contiene i comandi che possono essere invocati tramite un url, al fine di richiedere al sistema di eseguire una certa operazione; ogni comando individua una funzione personalizzata; è possibile indicare in quale gruppo di comandi si trova un comando.

L'ultimo aggregato, **proprietà**, è formato dalla sola tabella properties: in essa sono contenuti i valori delle variabili persistenti che lo sviluppatore può utilizzare nel codice. Un opportuno campo è stato inserito per indicare al sistema come codificare in database il formato dell'oggetto salvato.

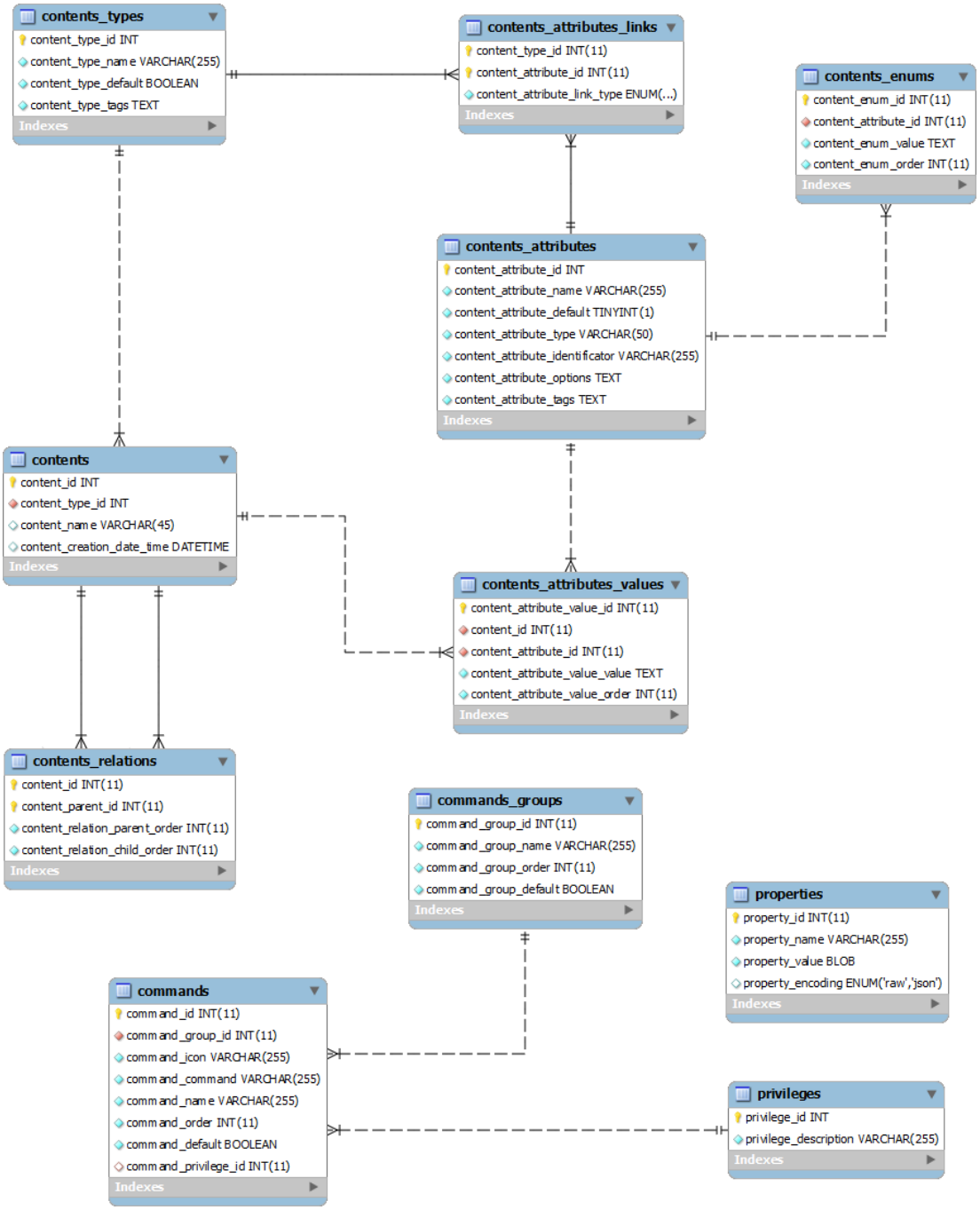


Figura 38 - Diagramma E-R della base di dati

3.3.2.1.1 Vincoli

I vincoli di chiave esterna così come le transazioni non sono presenti tra le funzionalità del motore predefinito MyISAM: una valida alternativa è l'utilizzo di un motore transazionale come InnoDB.

InnoDB permette il lock delle tabelle, permette di usare le transazioni e di inserire i vincoli di chiave esterna oltre ad essere più performante rispetto a MyISAM nel recupero dei dati e nell'immagazzinamento dei record: questi vengono ordinati in base al valore della chiave primaria e non in base all'inserimento.

Per garantire la consistenza delle informazioni contenute nella base di dati è necessario utilizzare gli strumenti messi a disposizione dal RDBMS, quali i vincoli di integrità referenziale. Questi si applicano indifferentemente ad ogni tupla della tabella di riferimento.

Si riportano i vincoli di integrità referenziale applicati, facendo riferimento alla Figura 38:

Tabella 1	Chiave esterna	Tabella 2	Chiave primaria	Aggiornamento	Rimozione
contents	content_type_id	contents_types	content_type_id	Cascata	Blocca
contents_attributes_links	content_type_id	contents_types	content_type_id	Cascata	Cascata
contents_attributes_links	content_attribute_id	contents_attributes	content_attribute_id	Cascata	Cascata
contents_enums	content_attribute_id	contents_attributes	content_attribute_id	Cascata	Cascata
contents_attributes_values	content_id	contents	content_id	Cascata	Cascata
contents_attributes_values	content_attribute_id	contents_attributes	content_attribute_id	Cascata	Nessuna azione
content_relations	content_id	contents	content_id	Cascata	Cascata
content_relations	content_parent_id	contents	content_id	Cascata	Nessuna azione
commands	command_group_id	commands_groups	command_group_id	Cascata	Cascata
commands	command_privilege_id	privileges	privilege_id	Cascata	Nessuna azione

Tabella 4 - Vincoli di integrità referenziale

La condizione **blocco** imposta tra la tabella contents e contents_types implica che la cancellazione di un tipo di contenuto (che permette di conoscere gli attributi e i valori che descrivono un contenuto) viene bloccata se esistono istanze di quel tipo.

La condizione **cascata** implica che, qualora il valore presente nel campo *chiave primaria* sia aggiornato o rimosso, la stessa operazione venga eseguita sulle tuple della tabella 1 che contengono il medesimo valore nel campo *chiave esterna*.

Il tool di sviluppo utilizzato per la progettazione e la realizzazione dello schema E-R illustrato in Figura 38 è MySQL Workbench. Le caratteristiche di questo strumento sono riassunte in Appendice C.

3.3.3 MDB2

Lo sviluppo di una applicazione, sia essa web-based o stand-alone, richiede l'utilizzo di una base di dati. Trattandosi di GestDevice, un'applicazione che trova applicazione in innumerevoli contesti aziendali, possiamo trovarci di fronte a vincoli tecnologici inaspettati: l'utilizzo di MySQL impone di utilizzare una macchina fornita di tale RDBMS. Non tutte le aziende dispongono di una licenza per questo prodotto e potrebbero non volerne una, sia per motivi economici che di accordi commerciali.

Considerando che il linguaggio SQL è uno standard per quanto riguarda l'interrogazione e l'aggiornamento dei dati contenuti nelle basi di dati, possiamo osservare cosa offre il mercato: PostgreSQL, Microsoft Access, Oracle, etc. Ognuno di questi utilizza una versione di SQL molto simile alle altre, ma introduce sempre un certo livello di personalizzazione della sintassi. Si pone allora il problema, durante la scrittura del codice sorgente di una applicazione, di utilizzare la sintassi maggiormente compatibile con tutte le altre, al fine di rendere portabile il sorgente.

Purtroppo non è possibile individuare la sintassi più compatibile, ma esistono, fortunatamente, librerie che permettono di specificare le query in un linguaggio intermedio che poi verrà tradotto nella sintassi corretta per il RDBMS scelto. Una libreria, largamente utilizzata in PHP, e disponibile su PEAR (PHP Extension and Application Repository), è MDB2.

MDB2 fornisce delle API che supportano indifferentemente qualsiasi RDBMS. La caratteristica che contraddistingue questa libreria dalle altre è che essa assicura la completa portabilità del codice SQL scritto.

Tra le altre caratteristiche riportiamo la gestione automatica delle condizioni di errore, il supporto alle transazioni, il lock delle tabelle e il controllo automatico del quoting dei parametri. Le query di GestDevice sono tutte scritte utilizzando il paradigma prepare & execute, descritto di seguito:

```
<?php
     /*
      * Execute prepared queries and return a resultset, otherwise
throw an exception.
      */
     function executePreparedQuery($string, array $data, $data_types
                                    = NULL, $op_type = "result")
     {
          $mdb2 = DB::getDbHandler();

          if (strtolower($op_type) === "manip")
               $op_type = MDB2_PREPARE_MANIP;
          else
               $op_type = MDB2_PREPARE_RESULT;

          $statement = $mdb2->prepare($string, $data_types, $op_type);
          if(PEAR::isError($statement))
               throw new Exception("Impossibile creare lo statement: "
               .$statement->getMessage() );
          $result = $statement->execute($data);
          if (PEAR::isError($result))
          {
               $str_data = debug($data, false, false);
               throw new Exception("La seguente query ha generato un
               errore non previsto: \"".$string."\" with data $str_data
               - ".$result->getMessage()." - \"".mysql_error()."\"");
          }
          return $result;
     }

     $sql = "
               INSERT INTO properties (property_value,
               property_encoding, property_name)
               VALUES (?, ?, ?);";
     }
     $data = array("valore", "codifica", "nome");
     $data_types = array('text', 'text', 'text');
     $result = DB::executePreparedQuery($sql, $data, $data_types,
     "manip");
?>
```

L'utilizzo dei segnaposto "*?*" permette di specificare la presenza di un valore da inserire. L'array dei valori, nell'esempio *$data*, contiene, nell'ordine di inserimento, i valori da sostituire ai segnaposto. L'array dei tipi dei valori, nell'esempio *$data_types*, consente di specificare, nello stesso ordine dell'array dei valori, i tipi dei dati indicati nell'array *$data*.

I dati da inserire devono essere specificati senza nessuna forma di quoting particolare, in quanto la protezione da SQL Injection è garantita da MDB2. In altre parole, il programmatore può scrivere il codice che implementa la logica senza preoccuparsi di proteggere il proprio codice da attacchi di tipo SQL Injection.

In Appendice C è possibile prendere visione di come installare ed utilizzare questa libreria.

3.3.4 Core

Come introdotto precedentemente, il Core di GestDevice mette a disposizione dello Sviluppatore alcune funzionalità a supporto dello sviluppo di nuove funzionalità.

La principale caratteristica del Core è il supporto per le variabili persistenti, oltre al supporto per l'invio sicuro di messaggi a gruppi di utenti.

3.3.4.1 Variabili persistenti: salvataggio e recupero di variabili tra istanze di esecuzione

GestDevice garantisce la possibilità di accesso alle informazioni contenute nel database da qualsiasi punto del codice sorgente. Molto spesso è necessario considerare condizioni che si verificano in istanze diverse di esecuzione di uno o più script.

Ad esempio, le informazioni relative ai log dei dispositivi devono sopravvivere all'esecuzione di un interprete, per permettere, ad esempio, di conteggiare quanti eventi di un certo tipo si sono verificati su un certo terminale.

Utilizzando le variabili persistenti è possibile memorizzare qualunque tipo di variabile disponibile in php: da semplici stringhe ad array molto complessi.

Le variabili persistenti sono disponibili in qualsiasi funzione personalizzata, semplicemente invocando i metodi statici riportati di seguito.

Ogni variabile viene riconosciuta tramite un identificativo che costituisce il primo parametro delle funzioni.

La funzione *getProperty()* permette di reperire, in un punto qualsiasi dello script, il valore della variabile persistente specificata. La sintassi di questa funzione è la seguente:

```
$valore = GestDevice::getProperty($nome_variabile);
```

La sua definizione nella classe GestDevice è riportata di seguito:

```
<?php
    Class GestDevice
    {
        // ...

        /*
         * Retrieves the property with the given name
         */
        public static function getProperty($name)
        {
            $sql = "
                SELECT property_value, property_encoding
                FROM properties as p
                WHERE p.property_name = ?;";
            $data = array($name);
            $data_types = array('text');
            $result = DB::executePreparedQueryAndGetAssocArray($sql,
            $data, $data_types);

            if(empty($result))
                throw new Exception("La proprietà $name non
                esiste.");

            return $result[0]['property_encoding']==="json" ?
            json_decode($result[0]['property_value']) :
            $result[0]['property_value'];
        }

        // ...
    }
?>
```

La funzione *setProperty()* permette di salvare il valore di una variabile all'interno della variabile persistente specificata. La sintassi di questa funzione è la seguente:

```
GestDevice::setProperty($nome_variabile, $valore);
```

La sua definizione nella classe GestDevice è riportata di seguito:

```
<?php
    Class GestDevice
    {
        // ...

        /*
         * Sets the property with the given name with the specified
value
         * Use "raw" encoding for binary data
         */
        public static function setProperty($name, $value, $encoding
= "")
        {
            $encoding = strtolower($encoding);
            if (!in_array($encoding, array("json","raw")))
            {
                if (is_array($value) || is_object($value))
                    $encoding = "json";
                else
                    $encoding = "raw";
            }

            try
            {
                GestDevice::getProperty($name);
                $sql = "
                    UPDATE properties SET property_value = ?,
                    property_encoding = ? WHERE property_name = ?;";

            }
            catch(Exception $e)
            {
                $sql = "
                    INSERT INTO properties (property_value,
                    property_encoding, property_name)
                    VALUES (?, ?, ?);";
            }
            $value = $encoding === "json" ? json_encode($value) :
$value;
            $data = array($value, $encoding, $name);
            $data_types = array('text', 'text', 'text');
            $result = DB::executePreparedQuery($sql, $data,
            $data_types, "manip");
        }

        // ...
    }
?>
```

Questo semplice meccanismo a due funzioni si appoggia sulla tabella "properties", già vista nel capitolo 3.3.2.1.

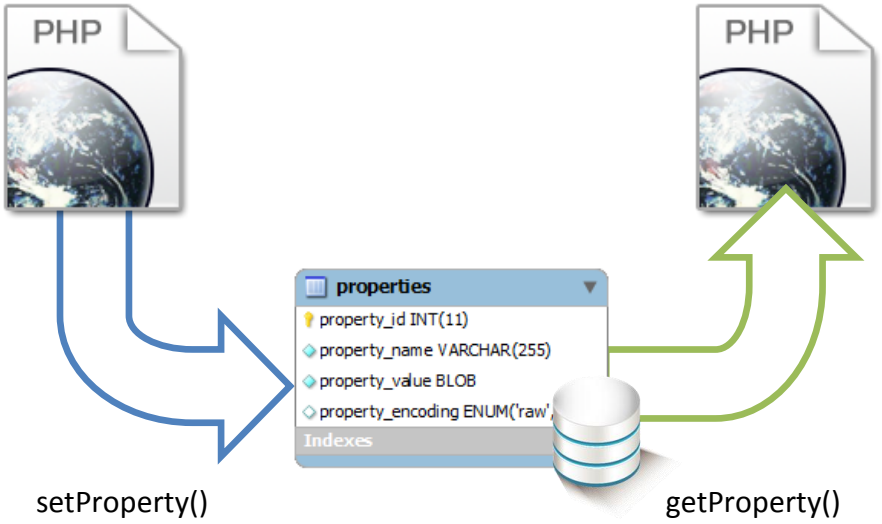


Figura 39 - Schema di utilizzo delle variabili persistenti

L'esecuzione di *getProperty()* si traduce nel lancio di una eccezione, qualora la proprietà specificata non sia settata. Si rende necessario l'utilizzo del paradigma try/catch:

```
<?php
    try
    {
        $x = GestDevice::getProperty("my_property_label");
    }
    catch(Exception $e)
    {
        $x = 0;
    }

    // Now you can use $x
    // ...
?>
```

3.3.4.2 *Invio di comunicazioni a gruppi di utenti*

Ogni sistema utilizza gli utenti come attori attivi che interagiscono con i dati in esso contenuti. Molto spesso è necessario comunicare a più soggetti il verificarsi di una certa condizione oppure semplicemente un promemoria: GestDevice mette a disposizione dello Sviluppatore una pratica funzionalità di invio di messaggi verso utenti definiti.

In particolare si sfrutta il concetto di gruppo di utenti, già introdotto nel capitolo 2.1.2.1: la funzione permette di inviare messaggi di posta elettronica specificando l'identificativo univoco del gruppo. Naturalmente, lo Sviluppatore ha accesso a questa informazione.

La sintassi è la seguente:

```
GestDevice::sendMail($group_id, $object, $message);
```

3.3.5 Sicurezza

Un'applicazione Web deve garantire sicurezza verso eventuali attacchi esterni provenienti da malintenzionati e, nel caso di GestDevice, è assolutamente necessaria l'adozione di politiche di sicurezza adeguate.

Un primo approccio consiste nell'utilizzo del protocollo HTTPS che viene utilizzato per garantire trasferimenti riservati di dati nel web, in modo da impedire intercettazioni dei contenuti che potrebbero essere effettuati tramite la tecnica del man in the middle. L'utilizzo di tale protocollo implica l'acquisto di un certificato, al fine di autenticare la reale identità del server web. L'entità di questo problema è minimo, in quanto i costi per l'acquisto di un certificato sono dell'ordine delle centinaia di Euro.

L'altro approccio, dall'interno, consiste nella memorizzazione di informazioni estremamente importanti, come la password di ogni utente, con una combinazione di un sale (in inglese, salt) e una regola di applicazione, la cui stringa risultante deve essere cifrata con una funzione irreversibile (di hash) come MD5 o SHA-1.

4. Il caso d'uso UniPOS

Il dispositivo comunemente conosciuto come POS assolve a funzioni di invio dati verso un collettore appositamente predisposto. Nella vita quotidiana ne abbiamo esperienza nel momento in cui finalizziamo un acquisto in qualsiasi punto vendita, mentre in ambito universitario la soluzione UniPOS fa uso di questi dispositivi nel processo di verbalizzazione degli esami. La soluzione attraverso il POS ha infatti sostituito la precedente procedura di compilazione degli statini.

Poiché gli appelli d'esame sono concentrati in alcuni periodi dell'anno, questi dispositivi vengono lasciati in disuso per settimane o addirittura mesi. Accade purtroppo che, nel momento in cui si ha la necessità di utilizzare un POS, si verificano alcuni fastidiosi problemi come batteria scarica, carta esaurita, guasti hardware, etc.

Attualmente il gruppo di sviluppatori che gestisce questo sistema utilizza molteplici soluzioni ad hoc per monitorare gli aspetti essenziali della vita di questi dispositivi. Recentemente si è sentita la necessità di disporre di uno strumento che consentisse un maggior controllo sullo stato dei vari POS distribuiti in tutto l'Ateneo pisano.

Per consentire una corretta gestione dei POS, il gruppo UniPOS necessita di una piattaforma per verificare costantemente ed in tempo reale il loro stato.

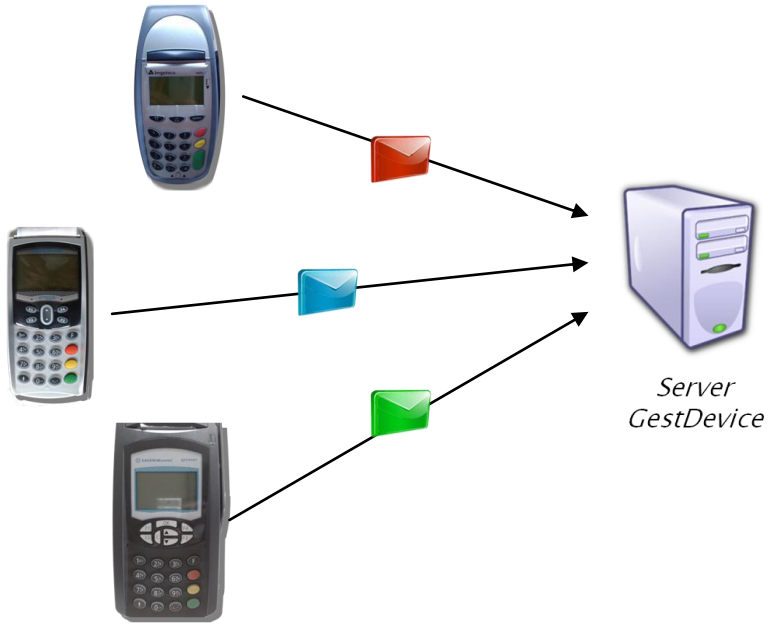


Figura 40 - Dispositivi POS inviano messaggi in differenti formati

I dispositivi POS sono equipaggiati con un software appositamente scritto per inviare notifiche sul proprio stato: il formato di queste notifiche varia in base al modello di dispositivo. Ciò non rappresenta assolutamente un problema per GestDevice, che è in grado di essere istruito alla comprensione di qualunque formato di messaggio. Nel capitolo 4.2.2 vediamo un esempio di messaggio inviato da un POS.

Da notare che i protocolli utilizzati per la trasmissione dei messaggi da parte dei dispositivi POS non sono legati a GestDevice, il quale si limita a lavorare sui messaggi in ingresso. In UniPOS è stato sviluppato un server che si occupa di recepire i messaggi dai vari dispositivi POS e di scrivere gli opportuni file su disco rigido.

La presenza di messaggi di diversi formati inviati dai POS non è l'unica caratteristica che ha portato all'introduzione di GestDevice nell'ambito di UniPOS. Infatti l'Ateneo pisano è organizzato a livelli:

- Facoltà
- Dipartimenti

- Uffici
- Postazioni
- ...

Come è stato ampiamente documentato nei capitoli precedenti, GestDevice è in grado di gestire e rappresentare al proprio interno una struttura organizzativa di questo tipo, al fine, ad esempio, di localizzare i dispositivi POS.

Uno dei requisiti per l'applicazione di GestDevice a UniPOS è proprio quello di poter localizzare i dispositivi all'interno dell'organizzazione. Nel capitolo che segue sono riassunti i requisiti di UniPOS.

Il sistema è stato realizzato per UniPOS come applicazione web: dato il grande numero di utenti dell'Università di Pisa che utilizzeranno GestDevice per UniPOS, occorrerebbe installare un applicativo su ogni macchina dell'Ateneo. Fortunatamente, nell'era di Internet e del Web 2.0, disponiamo di un applicazione estremamente potente, già preinstallata su ogni sistema operativo: il browser web. Utilizzando questa soluzione si risparmiano tempo e denaro connessi all'installazione del software, oltre che alla sua manutenzione, che implicherebbe una nuova installazione presso ogni macchina dell'Ateneo.

4.1 Requisiti

Il gestionale POS è un applicativo che permette di monitorare e gestire le informazioni concernenti i terminali POS, dal momento dell'acquisto fino a fine utilizzo.

In dettaglio il sistema si compone dei seguenti moduli:

- Catalogazione terminali
- Caratteristiche funzionali
- Configurazione POS
- Analisi statistica attività
- Verifiche e anomalie di sistema

- Guasti hardware

Ad alcune funzionalità è associato un sistema di generazione allarmi e promemoria per l'operatore.

4.1.1 Catalogazione terminali

L'amministrazione dei terminali necessita di un modulo che consenta di catalogare i terminali POS acquisiti. Questi rappresentano la fonte primaria che alimenta il sistema con le informazioni di base.

La scheda di un dispositivo si configura come un form di compilazione semi-guidata che permette di inserire i seguenti campi:

- numero identificativo univoco del terminale (è l'ID del POS di 8 cifre che individua univocamente il terminale stabilito dall'ateneo, potrebbe essere ad esempio il numero di inventario);
- numero seriale del terminale (campo numerico di 8 caratteri);
- data di acquisto terminale;
- data di attivazione terminale;
- tipologia firmware: dopo il primo inserimento, il dato può essere acquisito dinamicamente;
- configurazione caricata: dopo il primo inserimento, il dato può essere acquisito dinamicamente;
- dislocazione fisica: compilazione da menu a tendina (suddivisa in Facoltà, Dipartimento, struttura); campo non obbligatorio;
- indirizzo MAC (campo alfa numerico di 12 caratteri);
- indirizzo IP: acquisito dinamicamente e può variare ad ogni invio, in quanto legato alla base;
- manutenzione ordinaria e straordinaria: registra se il POS si trova in manutenzione;
- note (campo a digitazione libera): il contenuto del campo note deve essere inviato ad utenti definiti.

4.1.2 Caratteristiche funzionali

Partendo dalla locazione fisica di una postazione di invio, è necessario poter conoscere le caratteristiche funzionali dei terminali ivi allocati.

Il pannello presenta una tabella suddivisa per Facoltà, disposte in ordine alfabetico, con a fianco di ciascuna riportato il numero dei dispositivi di cui dispone, mentre in fondo alla tabella è riportato il numero totale.

Cliccando sul nome di una Facoltà si ottengono informazioni riportanti la suddivisione dei terminali rispetto ai Dipartimenti. Cliccando sul nome di un Dipartimento si entra nel livello di dettaglio superiore, ottenendo le informazioni sulla disposizione dei terminali all'interno delle sezioni/strutture del Dipartimento. Infine, cliccando sull'identificativo del terminale si ottengono le informazioni di base del terminale stesso.

In dettaglio verranno visualizzate almeno le seguenti informazioni che saranno acquisite dinamicamente:

- codice della configurazione installata (il dato sarebbe particolarmente utile nel caso in cui ci fosse una diversificazione in facoltà o aree);
- data ultimo aggiornamento effettuato con successo (data, ora, codice univoco del pacchetto dati caricato);
- data ultimo invio (data e ora dell'ultimo invio positivo);
- numero totale verbali inviati (numero verbali inviati nell'ultima operazione di invio con successo);
- totale esami inviati nel periodo di attività.

All'interno di questa funzionalità è presente un motore di ricerca che restituisce:

- all'inserimento dell'ID del POS, la localizzazione del terminale;
- all'inserimento del codice della configurazione, la quantità e il numero identificativo dei terminali che hanno tale configurazione installata.

4.1.3 Configurazione POS

Si rende necessaria la predisposizione di un modulo che contenga la storia delle versioni delle configurazioni installate sui POS, con la relativa data di emissione, ordinate dalla più recente (quella installata attualmente sui POS) a quella meno recente, e le informazioni sulle novità apportate a ciascun update.

4.1.4 Analisi statistica attività

Il sistema deve essere dotato di una funzione che permetta la generazione di analisi statistiche sulle attività del sistema UniPOS, sia nel suo complesso che per singolo terminale.

Il sistema gestisce la storicizzazione dei dati e consente di effettuare statistiche con impostazione di alcuni parametri.

Sono consentite le analisi statistiche rispetto a numero verbalizzazioni effettuate e inviate dal terminale.

Nel dettaglio sono consentite le analisi statistiche rispetto a:

- numero di verbalizzazioni effettuate e inviate dal terminale per data;
- frequenza di utilizzo;
- aggiornamenti effettuati relativi al pacchetto che il POS scarica.

Tutte le statistiche possono essere effettuate in forma grafica (attraverso grafici statistici) e in forma testuale ed è possibile stamparle.

Il modulo consente di definire alcune tipologie di alert come ad esempio la pianificazione di una verifica periodica di efficienza del terminale. In genere, trattandosi di dispositivi mobili, è utile impostare degli intervalli di tempo trascorsi i quali l'operatore esegue delle verifiche su consumo carta, eventuali danni visibili, etc.

4.1.5 Verifiche e anomalie di sistema

Per monitorare eventuali problemi software riscontrati sui dispositivi è necessario disporre di un modulo apposito, con notazione di:

- numero identificativo del terminale;
- tipologia dell'errore (è possibile visualizzare tutti gli errori che il POS è in grado di segnalare. In generale il sistema visualizzerà errori connessi a mancata possibilità di invio verbali, aggiornamento terminale, errore database);
- data dell'errore.

Permette di riportare alcune note tecniche utili alla risoluzione del guasto.

Permette l'invio di un'e-mail ad apposite mailing list:

- la sigla "POS-facoltà" nell'oggetto allo scopo di identificare agevolmente questa tipologia di messaggi;
- una breve stringa caratterizzante il tipo di problema;
- l'identificazione del POS;
- per segnalare eventuali interventi del personale tecnico o per suggerire eventuali azioni da intraprendere per ristabilire il normale e corretto funzionamento del sistema, senza interruzione del servizio.

La mail di destinazione è configurabile.

Le tipologie di errore segnalate sono:

- mancato aggiornamento dello specifico terminale;
- mancata generazione del pacchetto di configurazione;
- errore database;
- probabile necessità di cambiare il rotolino di carta sulla base del consumo di carta per singolo verbale.

All'interno di questa funzionalità è presente un motore di ricerca dedicato che permette di:

- risalire ad una specifica tipologia di errore;
- effettuare ricerche per data, o intervalli di tempo precisi.

4.1.6 Guasti hardware

Il modulo che gestisce i problemi hardware riscontrati sui dispositivi, con indicazione di:

- numero identificativo del terminale;
- tipologia dell'errore;
- data dell'errore.

All'interno di questa funzionalità è presente un motore di ricerca dedicato che permette di:

- risalire facilmente ad un certo guasto, tramite l'inserimento di parole chiave;
- di effettuare ricerche per data o lasso di tempo;
- di effettuare ricerche per tipologia di guasto.

4.1.7 Account Utenti

L'accesso al sistema è protetto e vengono memorizzate la generalità dell'utente che ha effettuato modifiche e inserimenti nel sistema.

All'interno del sistema è possibile individuare categorie diverse di utente, definite in base a diritti prestabiliti e aventi accesso a determinate funzionalità (sistema a livelli):

- un gruppo amministratore (di seguito denominato admin), avente competenza sull'intero sistema;
- un gruppo amministratore locale (di seguito denominato admin di Facoltà) con privilegi per inserire i dati relativi alla catalogazione e al controllo dello stato dei POS, per nominare gli utenti operatore;
- un gruppo di utenti operatore (di seguito denominato oper), ha visione solo sulle attività di gestione e manutenzione dei POS.

Gli admin hanno la visione globale del sistema e possono compiere le seguenti azioni:

- creare/eliminare utenti (admin, admin di Facoltà, oper);

- assegnare i diritti;
- inserire le informazioni;
- monitorare il sistema;
- rispondere agli alert.

Gli admin di Facoltà hanno la visione locale del sistema e possono compiere le seguenti azioni:

- creare/eliminare oper locali;
- assegnare i diritti agli oper;
- inserire le informazioni relativa alla propria area di Facoltà;
- monitorare il sistema;
- rispondere agli alert relativi alla propria area di competenza.

L'utente oper potrà eseguire le sole operazioni assegnategli in fase di creazione account (la configurazione dei diritti di un utente è modificabile solo dall'admin). In fase di creazione dell'account, l'admin deve indicare l'area di appartenenza dell'oper (possibilità di indicare più Facoltà).

4.2 Personalizzazione del sistema

Ogni contesto lavorativo ha esigenze diverse rispetto agli altri e, come tale, gli strumenti di supporto devono essere in grado di adattarsi. GestDevice non fa eccezione, facendo come suo punto di forza l'estrema flessibilità e personalizzabilità.

L'applicazione di GestDevice al contesto di lavoro di UniPOS ha richiesto varie personalizzazioni, in virtù delle richieste espresse nel capitolo 4.1. Nei capitoli che seguono sono illustrati gli interventi realizzati.

4.2.1 Tipologie di utenti

Data l'estensione fisica e organizzativa dell'Ateneo pisano, si rende necessario compartimentare l'accesso alle informazioni. In particolare è stata richiesta l'introduzione di un nuovo tipo di utente: l'Amministratore di Facoltà.

Questo soggetto è una persona interna alla facoltà di competenza, che può quindi gestire Operatori al di sotto di esso. Naturalmente occorre limitare il raggio di azione degli Amministratori di Facoltà, in quanto operanti in contesti diversi. Per questo motivo è stato introdotto un attributo per il tipo "utente" denominato "contesto" che si occupa di memorizzare il contesto di lavoro dell'utente.

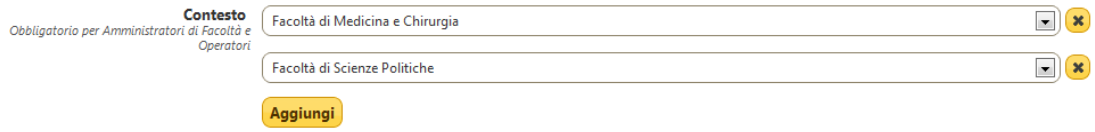


Figura 41 - Attributo "contesto" nel profilo utente

Selezionando una o più facoltà, l'utente si trova ad avere limitata giurisdizione solamente sui dispositivi collocati all'interno delle facoltà assegnategli.

Un Amministratore di Facoltà può creare, modificare o rimuovere utenti Operatore a piacimento: questi soggetti si trovano quindi a dipendere dal loro creatore, il quale può assegnare uno o più contesti operativi, tra quelli che possiede (es. Facoltà di Medicina e Chirurgia), e privilegi. Su questo ultimo aspetto si è incentrata una buona parte del lavoro di personalizzazione.

Analogamente a quanto detto per il "contesto", un Amministratore di Facoltà deve avere la possibilità di concedere privilegi ai propri sottoposti. Naturalmente il set di privilegi associabili è ristretto a quelli che egli stesso possiede, essendo comunque un utente subordinato alle figure di livello superiore (Amministratore, Sviluppatore, Super Amministratore).

4.2.2 Messaggi POS e interpreti

I dispositivi POS inviano messaggi verso il server UniPOS, il quale si occupa di riversarli su file, in un formato come quello che segue:

```
<?xml version="1.0"?>
<log-tag>
    <fields-tag nserial="11549755" firmware="2.2.0"
    date="20110415"/>
    <infos-tag ipaddr="192.168.0.2" nverbs="2"/>
    <infos-tag ultaggrn="15/04/2011" ultinvio="15/04/2011"
    esito="OK"/>
</log-tag>
```

Come si può notare il formato è XML ma la semantica del file è specifica. Per interpretare un file come questo, occorre istruire GestDevice affinché sia in grado di comprenderlo.

Si è resa necessaria la creazione di un interprete personalizzato. Di seguito riportiamo i punti cruciali del codice sorgente, tralasciando eventuali passaggi non essenziali alla trattazione del problema:

```
<?php
include_once(dirname(__FILE__)."/../class.interpreter.php");

Class POSXMLInterpreter extends Interpreter
{
    public static function canParse(&$string)
    {
        // Check if the given string can be parsed by this
        // interpreter
        $xml = new SimpleXMLElement($string);
        if (!$xml)
            return false;

        S// Search for <log-tag><infos-tag>
        $fields = $xml->xpath('/log-tag/fields-tag');
        $infos = $xml->xpath('/log-tag/infos-tag');
        if(!count($result) || !count($fields))
            return false;

        return true;
    }

    public static function parse(&$string)
    {
        // Parse the given string
        $xml = new SimpleXMLElement($string);
        $fields = $xml->xpath('/log-tag/fields-tag');
        $infos = $xml->xpath('/log-tag/infos-tag');

        foreach($fields as $node)
        {
            foreach($node->attributes() as $key => $value)
            {
                if(strtolower($key)==="nserial")
                {
                    $nserial = strtolower($value);
                    break;
                }
                // Get other values
                // ...
            }
        }

        $esito = "error";
        foreach($infos as $node)
```

```
        {
            foreach($node->attributes() as $key => $value)
            {
                if(strtolower($key)==="esito")
                {
                    $esito = strtolower($value);
                    break;
                }
                // Get other values
                // ...
            }
        }

        switch ($esito)
        {
            case "ok":
                // Do nothing
                break;
            case "error":
            default:
                if(!isset($nserial))
                    break;

                $var_name = "errors_" . $nserial;

                try
                {
                    $x = GestDevice::getProperty($var_name);
                }
                catch (Exception $e)
                {
                    $x = 0;
                }
                GestDevice::setProperty($var_name, ++$x);
                break;
        }
    }
}
?>
```

Il metodo *canParse()* verifica se è in grado di individuare un formato XML valido e, prima di ritornare un booleano che indica la sua capacità di leggere questo formato, verifica se nel file viene identificato lo schema composto da un tag *log-tag* come primo elemento e poi dei tag figli identificati da *infos-tag* o da *fields-tag*.

Qualora il sistema ottenga una risposta affermativa dall'interprete, invoca il metodo *parse()*. Questo si occupa di verificare, tra gli altri, la presenza dei tag necessari alla notifica dell'errore e memorizza i relativi valori nella variabili locali.

Una volta che ha identificato l'esito dell'operazione, effettua le opportune operazioni: in caso di errore, recupera la variabile che memorizza gli errori del

dispositivo con il numero seriale indicato, la incrementa e la memorizza nuovamente.

4.2.3 Attività programmate personalizzate

UniPOS necessita di un monitoraggio costante delle condizioni di errore o di avviso: questo non può essere delegato agli interpreti, che svolgono soltanto un ruolo di immissione dei dati nel sistema.

GestDevice mette a disposizione dello sviluppatore un pratico scheduler per eseguire processi personalizzati atti alla rilevazione di condizioni critiche e alla notifica all'utente di competenza.

L'interprete mostrato nel capitolo 4.2.2 si occupa di rilevare gli errori o le condizioni anomale che i dispositivi comunicano al sistema. Il passo successivo al rilevamento è quello di memorizzare le opportune variabili persistenti (vedi capitolo 3.3.4.1) per un successivo utilizzo.

Di seguito si riporta il codice sorgente di una attività programmata che si occupa di informare l'amministratore quando il numero di errori, di un qualsiasi dispositivo, raggiunge o supera il numero 3:

```
<?php
    // ...

    // List all POS devices
    $devices = Device::getAllDevices("pos");
    // ...
    foreach($devices as $dev)
    {
        if(isset($dev->parsed_attributes['serial_number']) && ($sn =
$dev->parsed_attributes['serial_number']))
        {
            $var_name = "errors_" . $sn;
            try
            {
                $errors = GestDevice::getProperty($var_name);
            }
            catch (Exception $e)
            {
                $errors = 0;
            }

            // Check numbers of errors
            if ($errors>2)
```

```
            {
                // Sends an email to the admin
                GestDevice::sendMail("admin@unipos.unipi.it",
"Errore POS", "Il POS con numero di serie '$sn' si trova in una
condizione di errore.");

                // Clear errors
                GestDevice::setProperty($var_name, 0);
            }
        }
    }
?>
```

Questo task viene eseguito alle ore 8 di mattina di ogni giorno. Ciò si traduce nella seguente sintassi cron:

```
0 8 * * *
```

In questo orario, questo script verifica che ogni dispositivo POS registrato nel sistema non abbia segnalato per più di 2 volte una condizione di errore. In caso negativo, lo script richiede l'invio di un messaggio di posta elettronica all'amministratore.

4.2.4 Assegnamento multiplo dei privilegi

Una delle funzionalità più interessanti che è stata aggiunta per personalizzare GestDevice per UniPOS è quella di assegnamento multiplo dei privilegi: una pratica interfaccia consente di selezionare molteplici privilegi da assegnare a molteplici utenti.

Figura 42 - Interfaccia di assegnamento multiplo dei privilegi

Disponendo di numerosi utenti e numerosi privilegi si è posto il problema di un rapido assegnamento di questi ultimi, senza ricorrere alla modifica di ogni profilo utente.

L'interfaccia presentata risulta estremamente semplice e consente, nota bene, di aggiungere privilegi agli utenti selezionati.

4.2.5 Localizzazione dispositivi

Come abbiamo avuto modo di vedere nei primi capitoli, per raggiungere una gestione efficiente dei dispositivi disponibili, occorre poterli localizzare con precisione e tempestività. Una volta specificata la locazione esatta di ogni dispositivo all'interno dei locali aziendali, si pone il problema di conoscere la posizione di uno di questi, a partire dal suo identificativo.

Per questo motivo è stata realizzata una nuova funzionalità che esplode la struttura organizzativa (Facoltà, Dipartimenti, etc.) per individuare tutti i dispositivi presenti.

Figura 43 - Funzionalità di localizzazione dispositivi

La funzionalità di ricerca è disponibile in alto, compilando un campo con l'identificativo preciso del dispositivo che si intende localizzare; una volta premuto il pulsante "Cerca" il sistema si mette alla ricerca del dispositivo richiesto e ne mostra l'ubicazione.

5. Conclusioni

L'attività di progettazione e sviluppo di GestDevice ha richiesto 4 mesi di lavorazione in cui sono state prese in considerazione tutte le problematiche di monitoraggio delle risorse aziendali.

Quando si cerca di realizzare uno strumento software il più possibile flessibile e configurabile, è necessario realizzare un trade-off tra il rischio di scivolare nell'ambito di un caso specifico e il rischio opposto di non riuscire a tracciare dei limiti all'eccessiva generalità.

GestDevice ha preso in considerazione qualsiasi dispositivo capace di inviare messaggi in qualunque formato. Le informazioni estratte da tali messaggi sono memorizzate in un database e costituiscono l'input per un set di funzioni di elaborazione predefinite alle quali possono essere aggiunte funzioni personalizzate. A livello organizzativo l'applicativo ha delineato profili di utenza tali da poter distinguere i possibili ruoli all'interno dell'azienda. Nell'ambito di ogni singolo profilo, GestDevice ha reso possibile l'attribuzione di un set di privilegi specifico per ogni utente al fine di restringerne l'accesso soltanto a determinate funzioni. Inoltre l'introduzione della possibilità di eseguire operazioni pianificate ha alleggerito il lavoro degli operatori che non sono più tenuti ad effettuare controlli manuali.

L'interfaccia grafica è stata curata in particolar modo seguendo linee guida estremamente rigide, non soltanto riguardo all'usabilità, ma anche alla semplicità con cui l'intero ambiente grafico può essere rivoluzionato in funzione delle esigenze della specifica azienda.

Nell'ambito universitario GestDevice ha trovato applicazione nel progetto UniPOS, l'innovativo sistema di verbalizzazione degli esami che ha mandato in pensione gli statini cartacei sostituendoli con dei dispositivi POS in grado registrare i risultati degli esami semplicemente strisciando le tessere degli studenti. Questi dispositivi, distribuiti tra i docenti dell'Ateneo, necessitano di essere rintracciati e controllati frequentemente, rendendo GestDevice lo strumento più adatto allo scopo.

GestDevice per UniPOS riporta l'intera struttura organizzativa dell'Ateneo fino al livello del singolo ufficio, o addirittura della singola postazione. Per delegare in modo ottimale la gestione dei dispositivi è perfino possibile assegnare una o più aree di competenza ad ogni utente, limitando il proprio raggio d'azione alle Facoltà assegnategli.

Per tutte le precedenti motivazioni e visti i risultati positivi ottenuti per UniPOS, GestDevice ha le potenzialità per diventare uno strumento di gestione adottabile in tutti quegli ambiti nei quali si fa uso di molteplici dispositivi che debbano essere monitorati costantemente, siano essi semplici POS, telefoni cellulari, smartphone o persino complessi macchinari industriali.

Appendice A Codice sorgente delle classi per la rappresentazione di oggetti

A.1 class.db.php

```
<?php

class DB
{
    /*
     * Return the MDB2 handler
     */
    static function getDbHandler()
    {
        $mdb2 = MDB2::singleton();
        if (PEAR::isError($mdb2))
            throw new Exception("Impossibile connettersi al
database: ".mysql_error()."\"");
        return $mdb2;
    }

    /*
     * Execute queries and return a resultset, otherwise throw an
exception.
     */
    static function executeQuery($string)
    {
        $mdb2 = DB::getDbHandler();
        $result = $mdb2->query($string);
        if (PEAR::isError($result))
            throw new Exception("La seguente query ha generato un
errore non previsto: \"".$string."\" - \"".mysql_error()."\"");
        return $result;
    }

    /*
     * Execute queries and return the number of rows
     */
    static function executeQueryAndGetNumRows($string)
    {
        $result = DB::executeQuery($string);
        return $result->numRows();
    }

    /*
     * Execute queries and return an associative array containing
"column_name" => "value" for each entry
     */
    static function executeQueryAndGetAssocArray($string)
    {
        $mdb2 = DB::getDbHandler();
        $result = $mdb2->queryAll($string, null,
MDB2_FETCHMODE_ASSOC);
        if (PEAR::isError($result))
            throw new Exception("La seguente query ha generato un
errore non previsto: \"".$string."\" - \"".mysql_error()."\"");
        return $result;
```

```
    }

    /*
     * Execute prepared queries and return a resultset, otherwise
throw an exception.
     */
    static function executePreparedQuery($string, array $data,
$data_types = NULL, $op_type = "result")
    {
        $mdb2 = DB::getDbHandler();

        if (strtolower($op_type) === "manip")
            $op_type = MDB2_PREPARE_MANIP;
        else
            $op_type = MDB2_PREPARE_RESULT;

        $statement = $mdb2->prepare($string, $data_types, $op_type);
        if(PEAR::isError($statement))
            throw new Exception("Impossibile creare lo statement: "
. $statement->getMessage() );
        $result = $statement->execute($data);
        if (PEAR::isError($result))
        {
            $str_data = debug($data, false, false);
            throw new Exception("La seguente query ha generato un
errore non previsto: \"".$string."\" with data $str_data -
".$result->getMessage()." - \"".mysql_error()."\"");
        }
        return $result;
    }

    /*
     * Execute prepared queries and return an associative array
containing "column_name" => "value" for each entry
     */
    static function executePreparedQueryAndGetAssocArray($string,
array $data, $data_types = NULL)
    {
        $result = DB::executePreparedQuery($string, $data,
$data_types);
        return $result->fetchAll(MDB2_FETCHMODE_ASSOC);
    }

    /*
     * Execute prepared queries and return the number of rows
     */
    static function executePreparedQueryAndGetNumRows($string, array
$data, $data_types = NULL)
    {
        $result = DB::executePreparedQuery($string, $data,
$data_types);
        return $result->numRows();
    }

    /*
     * Execute prepared queries and return the last inserted id
     */
```

```
    static function
executePreparedQueryAndGetLastInsertedId($string, array $data,
$data_types = NULL)
    {
        $mdb2 = MDB2::singleton();
        $result = DB::executePreparedQuery($string, $data,
$data_types, "manip");
        return $mdb2->lastInsertID();
    }

    /*
     * Disable auto commit
     */
    static function disableAutoCommit()
    {
        DB::executeQuery("SET autocommit=0;");
    }

    /*
     * Commits executed queries
     */
    static function commit()
    {
        DB::executeQuery("COMMIT;");
    }

    /*
     * Rollbacks executed queries
     */
    static function rollback()
    {
        DB::executeQuery("ROLLBACK;");
    }

    /*
     * Starts a new transaction
     */
    static function startTransaction()
    {
        DB::executeQuery("START TRANSACTION;");
    }
}

?>
```

A.2 class.content.php

```
<?php
include_once(realpath(dirname(__FILE__))."/class.instance.php");

/**
 *  This class maps every entry in the "contents" table
 */
class Content extends Instance
{
    static $instances = array();        // array of instances

    /**
```

```
     *  Implements get or create by ID
     */
    public static function findInstance($id, $data = NULL)
    {
        return parent::findClassInstance(get_class(),
array("content_id"=>$id), $data);
    }

    /**
     *  Return the existance of a row (by its id)
     */
    public static function exists(array $ids)
    {
        $q = "SELECT * FROM contents as c WHERE 1";
        $data = array();
        foreach($ids as $key => $value)
        {
            $q.= " AND c.`".$key."` = ?";
            $data[] = $value;
        }
        $q.= " LIMIT 1;";
        $num = DB::executePreparedQueryAndGetNumRows($q, $data);
        return ($num>0) ? true : false;
    }

    /**
     *  Instances a new object starting by its own id or, if
$data!=NULL, by the $data
     */
    public function __construct($id, $data=NULL)
    {
        $classname = get_class($this);

        // add new instance to static vector
        $classname::$instances[] = $this;

        if(is_null($data))
        {
            $sql = "
                SELECT *
                FROM contents as c
                WHERE c.content_id = ?
                LIMIT 1;";
            $data =
DB::executePreparedQueryAndGetAssocArray($sql,array($id));
            if (count($data))
                $data = $data[0];
            else
                throw new Exception("Impossibile trovare content con
id $id.");
        }

        $this->update($data);
    }

    function update($data = NULL)
    {
        // clear actual properties and magic properties
        if(isset($this->content_attribute_values))
```

```
        {
            foreach($this->content_attribute_values as $av)
            {
                $av->update();
            }
        }

        foreach($this as $prop => $val)
            if($prop != 'content_id')
                unset($this->$prop);

        if(is_null($data))
        {
            $sql = "
                SELECT *
                FROM contents as c
                WHERE c.content_id = ?
                LIMIT 1;";
            $data =
DB::executePreparedQueryAndGetAssocArray($sql,array($this-
>content_id));
            if (count($data))
                $data = $data[0];
            else
                throw new Exception("Impossibile trovare content con
id $id.");
        }

        if (count($data))
        {
            // create attributes
            foreach($data as $key => $item)
                $this->$key = $item;

            // create attributes array
            $this->attributes = array();
            foreach($this->content_attribute_values as $item)
            {
                $identifier = $item->content_attribute-
>content_attribute_identificator;
                $this->attributes[$identifier][] = $item;
            }

            // create parsed attributes
            $this->parsed_attributes = array(
                'content_id' => $this->content_id
            );
            foreach($this->attributes as $identifier => $items)
            {
                foreach($items as $ref)
                {
                    if(!is_null($ref->content_attribute->plugin))
                    {
                        $ref->content_attribute->plugin-
>parseValue($this->parsed_attributes[$identifier], $ref);
                    }
                }
            }
        }
```

```
    }

    public function __get($name)
    {
        switch($name)
        {
            case "content_type":
                return $this->content_type =
ContentType::findInstance($this->content_type_id);
                break;
            case "content_attribute_values":
                $this->content_attribute_values =
ContentAttributeValue::searchByContentId($this->content_id);
                return $this->content_attribute_values;
                break;
            case "content_relation_parents":
                return $this->content_relation_parents =
ContentRelation::searchByContentId($this->content_id);
                break;
            case "content_relation_children":
                return $this->content_relation_children =
ContentRelation::searchByContentParentId($this->content_id);
                break;
            case 'descendent_ids':
                $arr = array((int)$this->content_id);
                foreach($this->content_relation_children as $rel)
                {
                    $arr2 = $rel->content->descendent_ids;
                    $arr = array_merge($arr, $arr2);
                }
                return $this->descendent_ids = $arr;
                break;
        }
        trigger_error("Undefined property '$name' in " .
get_class($this));
    }

    /*
     * Creates a new content and returns the new instanced
content_id
     * In case of errors, returns false
     */
    public static function createContent($content_name,
$content_type_id)
    {
        $sql = "
            INSERT INTO contents
            (content_id, content_type_id, content_name,
content_creation_date_time)
            VALUES
            (NULL, ?, ?, NOW());";
        $data = array($content_type_id, $content_name);
        $data_types = array('integer', 'text');
        try
        {
            $inserted_id =
DB::executePreparedQueryAndGetLastInsertedId($sql, $data,
$data_types);
        }
```

```
        catch (Exception $e)
        {
            return false;
        }
        return $inserted_id;
    }

    /*
     * Modify content
     * In case of errors, returns false, otherwise true
     */
    public function modify($content_name)
    {
        try
        {
            if(!Content::exists(array('content_id' => $this-
>content_id)))
                throw new Exception("Il contenuto da modificare non
esiste.");

            $sql = "
                UPDATE contents SET
                content_name = ?
                WHERE content_id = ?
                LIMIT 1;";
            $data = array($content_name, $this->content_id);
            $data_types = array('text', 'integer');

            DB::executePreparedQuery($sql, $data, $data_types,
"manip");

            $this->update();
        }
        catch (Exception $e)
        {
            return false;
        }
        return true;
    }

    /*
     * Delete the content with the given content id
     * Returns true (in case of success) or false (otherwise)
     */
    public static function deleteContent($content_id)
    {
        if (!Content::exists(array("content_id" => $content_id)))
            return false;

        $sql = "
            DELETE FROM contents
            WHERE content_id = ?;";
        $data = array($content_id);
        $data_types = array('integer');
        try
        {
            DB::executePreparedQuery($sql, $data, $data_types,
"manip");
        }
```

```
        catch (Exception $e)
        {
            return false;
        }
        return true;
    }
}

?>
```

A.3 class.content_attribute.php

```
<?php
include_once(realpath(dirname(__FILE__))."/class.instance.php");

/**
 *  This class maps every entry in the "contents_attributes" table
 */
class ContentAttribute extends Instance
{
    static $instances = array();        // array of instances

    static $type_managers = array();

    // keep trace if type managers have been loaded
    static $loaded = false;

    public static function addTypeManager($type_manager_name)
    {
        ContentAttribute::$type_managers[] = $type_manager_name;
    }

    public static function addDefaultTypeManagers()
    {

foreach(glob(realpath(dirname(__FILE__))."/attribute_type_managers/*
.php") as $file)
        {
            $content = file_get_contents($file);
            $matches = array();
            if(preg_match_all("/class\\s{1,}([_a-z]{1,}[_a-z0-
9]{0,})\s+extends\s+AttributeTypeManager\s{0,}[{]/i", $content,
$matches))
            {
                include_once($file);
                foreach($matches[1] as $classname)
                    ContentAttribute::addTypeManager($classname);
            }
        }

        ContentAttribute::$loaded = true;
    }

    /*
     * Decides which type manager use to interpret the attribute
     */
    public static function findTypeManager(ContentAttribute
$attribute)
```

```
    {
        // Loads type managers
        if (!ContentAttribute::$loaded)
            ContentAttribute::addDefaultTypeManagers();

        foreach(ContentAttribute::$type_managers as $type_manager)
        {
            if($type_manager::canHandle($attribute))
                return $type_manager;
        }
        throw(new Exception("Plugin not found for type:
'".$attribute->content_attribute_type . "'"));
    }

    /*
     * Returns an array containing all allowed types and options
     */
    public static function allowedTypes()
    {
        $arr = array();
        foreach(ContentAttribute::$type_managers as $type_name)
            $type_name::allowedTypes($arr);
        return $arr;
    }

    /**
     *  Implements get or create by ID
     */
    public static function findInstance($id, $data = NULL)
    {
        return parent::findClassInstance(get_class(),
array("content_attribute_id"=>$id), $data);
    }

    /**
     *  Return the existance of a row (by its id)
     */
    public static function exists(array $ids)
    {
        $q = "SELECT * FROM contents_attributes as ca WHERE 1";
        foreach($ids as $key => $value)
        {
            $q.= " AND ca.`".$key."` = ?";
            $data[] = $value;
        }
        $q.= " LIMIT 1;";
        $num = DB::executePreparedQueryAndGetNumRows($q, $data);
        return ($num>0) ? true : false;
    }

    /*
     * Finds a content attribute by its own identificator and
returns the object
     */
    public static function getByIdentificator($identificator)
    {
        $sql = "
            SELECT content_attribute_id
            FROM contents_attributes as ca
```

```
            WHERE ca.content_attribute_identificator = ?
            LIMIT 1;";
        $data = array($identificator);
        $data_types = array('text');
        $result = DB::executePreparedQueryAndGetAssocArray($sql,
$data, $data_types);

        if(count($result) != 1)
            return NULL;

        return
ContentAttribute::findInstance($result[0]['content_attribute_id']);
    }

    /**
     *  Instances a new object starting by its own id or, if
$data!=NULL, by the $data.
     */
    public function __construct($id,$data=NULL)
    {
        $classname = get_class($this);

        // Add new instance to static vector
        $classname::$instances[] = $this;

        // Query DB if nothing is passed
        if(is_null($data))
        {
            $sql = "
                SELECT *
                FROM contents_attributes as ca
                WHERE ca.content_attribute_id = ?
                LIMIT 1;";
            $data = DB::executePreparedQueryAndGetAssocArray($sql,
array($id));
            if (count($data))
                $data = $data[0];
            else
                throw new Exception("Impossibile trovare
content_attribute con id $id.");
        }
        if (count($data))
        {
            foreach($data as $key => $item)
                $this->$key = $item;

            $this->options = array();
            // Parse options to create an array of options
            if (!is_null($this->content_attribute_options))
            {
                parse_str($this->content_attribute_options,$this-
>options);
                ksort($this->options);
            }

        }
    }

    public function __get($name)
```

```
    {
        switch($name)
        {
            case "content_enums":
                if ($this->content_attribute_type=='enum' || $this-
>content_attribute_type=='enum_multiple')
                    return $this->content_enums =
ContentEnum::searchByContentAttributeId($this-
>content_attribute_id);
                break;
            case "content_attribute_links":
                return $this->content_attribute_links =
ContentAttributeLink::searchByContentAttributeId($this-
>content_attribute_id);
                break;
            case "content_attribute_values":
                return $this->content_attribute_values =
ContentAttributeValue::searchByContentAttributeId($this-
>content_attribute_id);
                break;
            case "plugin":
                $p = ContentAttribute::findTypeManager($this);
                if(!is_null($p))
                   return $this->plugin = new $p($this);
                return NULL;
                break;
            case 'description':
                $cn = get_class($this->plugin);
                $types = array();
                $cn::allowedTypes($types);
                $types = $types[$this->content_attribute_type];
                foreach($types as $type)
                {
                    if($type['options'] === $this-
>content_attribute_options)
                    {
                        if(isset($type['description']))
                            return $this->description =
$type['description'];
                        else
                            break;
                    }
                }
                return $this->description = "";

        }
        trigger_error("Undefined property '$name' in " .
get_class($this));
    }


    /**
     *  Return an array of instances filtering by the given tag
     */
    public static function searchByTag($tag)
    {
        $sql = "
            SELECT *
            FROM contents_attributes as ca
```

```
            WHERE CONCAT(',', ca.content_attribute_tags, ',') LIKE
?;";
        $data = DB::executePreparedQueryAndGetAssocArray($sql,
array("%,$tag,%"));
        $return_data = array();
        foreach ($data as $row)
        {
            $return_data[] =
ContentAttribute::findInstance($row['content_attribute_id'], $row);
        }
        return $return_data;
    }

}

?>
```

A.4 class.content_attribute_link.php

```
<?php
include_once(dirname(__FILE__)."/class.instance.php");

/**
 *  This class maps every entry in the "contents_attributes_links"
table
 */
Class ContentAttributeLink extends Instance
{
    static $instances = array();        // array of instances

    /**
     *  Implements get or create by ID
     */
    public static function findInstance($attribute_id, $type_id,
$data = NULL)
    {
        return parent::findClassInstance(get_class(),
array("content_attribute_id" => $attribute_id,"content_type_id" =>
$type_id), $data);
    }

    /**
     *  Return the existance of a row (by its id)
     */
    public static function exists(array $ids)
    {
        $q = "SELECT * FROM contents_attributes_links as cal WHERE
1";
        $data = array();
        foreach($ids as $key => $value)
        {
            $q.= " AND cal.`".$key."` = ?";
            $data[] = $value;
        }
        $q.= " LIMIT 1;";
        $num = DB::executePreparedQueryAndGetNumRows($q, $data);
        return ($num>0) ? true : false;
    }
```

```
    /**
     *  Return an array of instances
     */
    public static function searchByContentTypeId($id)
    {
        $sql = "
            SELECT *
            FROM contents_attributes_links as cal
            WHERE cal.content_type_id = ? ORDER BY
cal.content_attribute_id ASC;";
        $data = DB::executePreparedQueryAndGetAssocArray($sql,
array($id));
        $return_data = array();
        foreach ($data as $row)
        {
            $return_data[] =
ContentAttributeLink::findInstance($row['content_attribute_id'],
$row['content_type_id'], $row);
        }
        return $return_data;
    }

    /**
     *  Return an array of instances
     */
    public static function searchByContentAttributeId($id)
    {
        $sql = "
            SELECT *
            FROM contents_attributes_links as cal
            WHERE cal.content_attribute_id = ? ORDER BY
cal.content_attribute_id ASC;";
        $data = DB::executePreparedQueryAndGetAssocArray($sql,
array($id));
        $return_data = array();
        foreach ($data as $row)
        {
            $return_data[] =
ContentAttributeLink::findInstance($row['content_attribute_id'],
$row['content_type_id'], $row);
        }
        return $return_data;
    }

    /**
     *  Instances a new object starting by its own id or, if
$data!=NULL, by the $data
     */
    public function __construct(array $ids,$data=NULL)
    {
        $classname = get_class($this);

        // add new instance to static vector
        $classname::$instances[] = $this;

        if(is_null($data))
        {
            $sql = "
```

```
                SELECT *
                FROM contents_attributes_links as cal
                WHERE 1";
            $values = array();
            foreach($ids as $key => $value)
            {
                //$q.= " AND cal.`".$key."` = '".$value."'";
                $q.= " AND cal.`".$key."` = ?";
                $values[] = $value;
            }
            $q.= " LIMIT 1;";
            $data = DB::executePreparedQueryAndGetAssocArray($sql,
$values);
            if (count($data))
                $data = $data[0];
            else
                throw new Exception("Impossibile trovare
content_attribute_link con id $id.");
        }

        if (count($data))
        {
            foreach($data as $key => $item)
                $this->$key = $item;
        }
    }

    public function isMandatory()
    {
        return ($this->content_attribute_link_type === "mandatory");
    }

    public function isSkeleton()
    {
        return ($this->content_attribute_link_type === "mandatory")
||
                ($this->content_attribute_link_type === "skeleton");
    }

    public function __get($name)
    {
        switch($name)
        {
            case "content_attribute":
                return $this->content_attribute =
ContentAttribute::findInstance($this->content_attribute_id);
                break;
            case "content_type":
                return $this->content_type =
ContentType::findInstance($this->content_type_id);
                break;
        }
        trigger_error("Undefined property '$name' in " .
get_class($this));
    }

    /*
     * Creates a new content attribute link and returns keys values
     * In case of errors, returns false
```

```
     */
    public static function create($content_type_id,
$content_attribute_id, $link_type)
    {
        $sql = "
            INSERT INTO contents_attributes_links
            (content_type_id, content_attribute_id,
content_attribute_link_type)
            VALUES
            (?, ?, ?);";
        $data = array($content_type_id, $content_attribute_id,
$link_type);
        $data_types = array('integer', 'integer', 'text');
        try
        {
            DB::executePreparedQueryAndGetLastInsertedId($sql,
$data, $data_types);
        }
        catch (Exception $e)
        {
            return false;
        }
        return array($content_type_id, $content_attribute_id);
    }

    /*
     * Delete the content attribute link
     * Returns true (in case of success) or false (otherwise)
     */
    public function delete()
    {
        $sql = "
            DELETE FROM contents_attributes_links
            WHERE content_type_id = ? AND content_attribute_id =
?;";
        $data = array($this->content_type_id, $this-
>content_attribute_id);
        $data_types = array('integer', 'integer');
        try
        {
            DB::executePreparedQuery($sql, $data, $data_types,
"manip");
        }
        catch (Exception $e)
        {
            return false;
        }
        return true;
    }
}
?>
```

A.5 class.content_attribute_value.php

```
<?php
include_once(realpath(dirname(__FILE__))."/class.instance.php");

/**
```

```
 *  This class maps every entry in the "contents_attributes_values"
table
 */
class ContentAttributeValue extends Instance
{
    static $instances = array();        // array of instances

    /**
     *  Implements get or create by ID
     */
    public static function findInstance($id, $data = NULL)
    {
        return parent::findClassInstance(get_class(),
array("content_attribute_value_id"=>$id), $data);
    }

    /**
     *  Return the existance of a row (by its id)
     */
    public static function exists(array $ids)
    {
        $q = "SELECT * FROM contents_attributes_values as cav WHERE
1";
        foreach($ids as $key => $value)
        {
            $q.= " AND cav.`".$key."` = ?";
            $data[] = $value;
        }
        $q.= " LIMIT 1;";
        $num = DB::executePreparedQueryAndGetNumRows($q, $data);
        return ($num>0) ? true : false;
    }

    /**
     *  Return an array of instances
     */
    public static function searchByContentAttributeId($id)
    {
        $sql = "
            SELECT *
            FROM contents_attributes_values as cav
            WHERE cav.content_attribute_id = ?;";
        $data = DB::executePreparedQueryAndGetAssocArray($sql,
array($id));
        $return_data = array();
        foreach ($data as $row)
        {
            $return_data[] =
ContentAttributeValue::findInstance($row['content_attribute_value_id
'], $row);
        }
        return $return_data;
    }

    /**
     *  Return an array of instances
     */
    public static function searchByContentId($id)
    {
```

```
        $sql = "
            SELECT *
            FROM contents_attributes_values as cav
            WHERE cav.content_id = ?
            ORDER BY cav.content_attribute_value_order ASC;";
        $data = DB::executePreparedQueryAndGetAssocArray($sql,
array($id));
        $return_data = array();
        foreach ($data as $row)
        {
            $return_data[] =
ContentAttributeValue::findInstance($row['content_attribute_value_id
'], $row);
        }
        return $return_data;
    }

    /**
     *  Instances a new object starting by its own id or, if
$data!=NULL, by the $data
     */
    public function __construct($id,$data=NULL)
    {
        $classname = get_class($this);

        // Add new instance to static vector
        $classname::$instances[] = $this;

        if(is_null($data))
        {
            $sql = "
                SELECT *
                FROM contents_attributes_values as cav
                WHERE cav.content_attribute_value_id = ?
                LIMIT 1;";
            $data = DB::executePreparedQueryAndGetAssocArray($sql,
array($id));
            if (count($data))
                $data = $data[0];
            else
                throw new Exception("Impossibile trovare
content_attribute_value con id $id.");
        }

        $this->update($data);
    }

    public function update($data = NULL)
    {
        foreach($this as $prop => $val)
            if($prop != 'content_attribute_value_id')
                unset($this->$prop);

        if(is_null($data))
        {
            $sql = "
                SELECT *
                FROM contents_attributes_values as cav
                WHERE cav.content_attribute_value_id = ?
```

```
                LIMIT 1;";
            $data = DB::executePreparedQueryAndGetAssocArray($sql,
array($this->content_attribute_value_id));
            if (count($data))
                $data = $data[0];
        }

        if (count($data))
        {
            foreach($data as $key => $item)
                $this->$key = $item;

            // Loading specific data through the correct plugin
            $this->content_attribute->plugin-
>objectConstructor($this);
        }
    }

    public function __get($name)
    {
        switch($name)
        {
            case "content_attribute":
                return $this->content_attribute =
ContentAttribute::findInstance($this->content_attribute_id);
                break;
            case "content":
                return $this->content = Content::findInstance($this-
>content_id);
                break;
        }

        trigger_error("Undefined property '$name' in " .
get_class($this));
    }

    /*
     * Creates a new content attribute value and returns the new
instanced content attribute value id
     * In case of errors, returns false
     * If order is not specified (or is null) the order will be set
automatically
     */
    public static function createAttributeValue($content_id,
$content_attribute_id, $value, $order = NULL)
    {
        // Order automatically set if null is passed
        if(is_null($order))
        {
            $sql = "
                INSERT INTO contents_attributes_values
                (content_attribute_value_id, content_id,
content_attribute_id, content_attribute_value_value,
content_attribute_value_order)
                SELECT NULL, ?, ?, ?,
(max(cav.content_attribute_value_order) + 1)
                FROM contents_attributes_values as cav
                WHERE cav.content_id = ? AND
cav.content_attribute_id = ?
```

```
                ;";
            $data = array($content_id, $content_attribute_id,
$value, $content_id, $content_attribute_id);
            $data_types = array('integer', 'integer', 'text',
'integer', 'integer');
        }
        else
        {
            $sql = "
                INSERT INTO contents_attributes_values
                (content_attribute_value_id, content_id,
content_attribute_id, content_attribute_value_value,
content_attribute_value_order)
                VALUES
                (NULL, ?, ?, ?, ?);";
            $data = array($content_id, $content_attribute_id,
$value, (int)$order);
            $data_types = array('integer', 'integer', 'text',
'integer');
        }
        try
        {
            $inserted_id =
DB::executePreparedQueryAndGetLastInsertedId($sql, $data,
$data_types);
        }
        catch (Exception $e)
        {
            return false;
        }
        return $inserted_id;
    }

    /*
     * Modify the current content attribute value
     * In case of errors, returns false
     */
    public function modify($value)
    {
        $sql = "
            UPDATE contents_attributes_values
            SET content_attribute_value_value = ?
            WHERE content_attribute_value_id = ?
            ;";
        $data = array($value, $this->content_attribute_value_id);
        $data_types = array('text', 'integer');
        try
        {
            DB::executePreparedQuery($sql, $data, $data_types,
"manip");

            $this->update();
        }
        catch(Exception $e)
        {
            return false;
        }
        return true;
    }
```

```
}
?>
```

A.6 class.content_type.php

```
<?php
include_once(dirname(__FILE__)."/class.instance.php");

/**
 *  This class maps every entry in the "contents_types" table
 */
Class ContentType extends Instance
{
    static $instances = array();        // array of instances

    /**
     *  Implements get or create by ID
     */
    public static function findInstance($id, $data = NULL)
    {
        return parent::findClassInstance(get_class(),
array("content_type_id"=>$id), $data);
    }

    /**
     *  Return the existance of a row (by its id)
     */
    public static function exists(array $ids)
    {
        $q = "SELECT * FROM contents_types as ct WHERE 1";
        $data = array();
        foreach($ids as $key => $value)
        {
            $q.= " AND ct.`".$key."` = ?";
            $data[] = $value;
        }
        $q.= " LIMIT 1;";
        $num = DB::executePreparedQueryAndGetNumRows($q, $data);
        return ($num>0) ? true : false;
    }

    /**
     *  Instances a new object starting by its own id or, if
$data!=NULL, by the $data
     */
    public function __construct($id,$data=NULL)
    {
        $classname = get_class($this);

        // Add new instance to static vector
        $classname::$instances[] = $this;

        if(is_null($data))
        {
            $sql = "
                SELECT *
                FROM contents_types as ct
```

```
                WHERE ct.content_type_id = ?
                LIMIT 1;";
            $data = DB::executePreparedQueryAndGetAssocArray($sql,
array($id));
            if (count($data))
                $data = $data[0];
            else
                throw new Exception("Impossibile trovare
content_type con id $id.");
        }

        $this->update($data);
    }

    function update($data = NULL)
    {
        if(is_null($data))
        {
            $sql = "
                SELECT *
                FROM contents_types as ct
                WHERE ct.content_type_id = ?
                LIMIT 1;";
            $data = DB::executePreparedQueryAndGetAssocArray($sql,
array($this->content_type_id));
            if (count($data))
                $data = $data[0];
        }

        if (count($data))
        {
            foreach($data as $key => $item)
                $this->$key = $item;
        }
    }

    public function __get($name)
    {
        switch($name)
        {
            case "contents":
                return $this->contents =
Content::searchByContentTypeId($this->content_type_id);
                break;
            case "content_attribute_links":
                return $this->content_attribute_links =
ContentAttributeLink::searchByContentTypeId($this->content_type_id);
                break;
        }
        trigger_error("Undefined property '$name' in " .
get_class($this));
    }

    public function getMandatoryAttributeLinks()
    {
        $sql = "
            SELECT cal.*
            FROM contents_attributes_links AS cal
            LEFT JOIN contents_attributes AS ca
```

```
            ON cal.content_attribute_id = ca.content_attribute_id
            WHERE cal.content_attribute_link_type IN ('mandatory')
AND
                  cal.content_type_id = ?
            ORDER BY ca.content_attribute_name ASC";
        $datas = DB::executePreparedQueryAndGetAssocArray($sql,
array($this->content_type_id));
        $arr = array();
        foreach($datas as $data)
            $arr[] =
ContentAttributeLink::findInstance($data['content_attribute_id'],
$data['content_type_id'], $data);
        return $arr;
    }

    public function getSkeletonAttributeLinks()
    {
        $sql = "SELECT cal.* FROM contents_attributes_links AS cal
                LEFT JOIN contents_attributes AS ca
                ON cal.content_attribute_id =
ca.content_attribute_id
                WHERE cal.content_attribute_link_type IN
('mandatory','skeleton') AND
                      cal.content_type_id = ?
                ORDER BY ca.content_attribute_name ASC";
        $datas = DB::executePreparedQueryAndGetAssocArray($sql,
array($this->content_type_id));
        $arr = array();
        foreach($datas as $data)
            $arr[] =
ContentAttributeLink::findInstance($data['content_attribute_id'],
$data['content_type_id'], $data);
        return $arr;
    }

    public function getAvailableAttributeLinks($filters = array())
    {
        $sql = "
            SELECT cal.*
            FROM contents_attributes_links AS cal
            LEFT JOIN contents_attributes AS ca
            ON cal.content_attribute_id = ca.content_attribute_id
            WHERE cal.content_type_id = ?";
        $data = array($this->content_type_id);
        $data_types = array('integer');
        if(is_array($filters) && count($filters)>0)
        {
            $no_ids = "\"" . implode("\", \"", $filters) . "\"";
            $sql .= " AND ca.content_attribute_identificator NOT IN
($no_ids)";
        }
        $sql .= " ORDER BY ca.content_attribute_name ASC";

        $datas = DB::executePreparedQueryAndGetAssocArray($sql,
$data, $data_types);
        $arr = array();
        foreach($datas as $data)
```

```
            $arr[] =
ContentAttributeLink::findInstance($data['content_attribute_id'],
$data['content_type_id'], $data);
        return $arr;
    }

    /*
     * Creates a new content type and returns the new instanced
content_type_id
     * In case of errors, returns false
     */
    public static function create($content_type_name, $default = 0)
    {
        $sql = "
            INSERT INTO contents_types
            (content_type_id, content_type_name,
content_type_default)
            VALUES
            (NULL, ?, ?);";
        $data = array($content_type_name, $default);
        $data_types = array('text', 'integer');
        try
        {
            $inserted_id =
DB::executePreparedQueryAndGetLastInsertedId($sql, $data,
$data_types);
        }
        catch (Exception $e)
        {
            return false;
        }
        return $inserted_id;
    }

    /*
     * Finds a content type by its own name and returns the object
     */
    public static function getByName($name)
    {
        $sql = "
            SELECT content_type_id
            FROM contents_types as ct
            WHERE ct.content_type_name = ?
            LIMIT 1;";
        $data = array($name);
        $data_types = array('text');
        $result = DB::executePreparedQueryAndGetAssocArray($sql,
$data, $data_types);

        if(count($result) != 1)
            return NULL;

        return
ContentType::findInstance($result[0]['content_type_id']);
    }

    /*
     * Modify content type
     * In case of errors, returns false, otherwise true
```

```
     */
    public function modify($content_type_name, $default = null)
    {
        try
        {
            if(!ContentType::exists(array('content_type_id' =>
$this->content_type_id)))
                throw new Exception("Il tipo di contenuto da
modificare non esiste.");

            $sql = "
                UPDATE contents_types SET
                content_type_name = ?".
                (!is_null($default)?", content_type_default =
?":"").""
                WHERE content_type_id = ?
                LIMIT 1;";
            if(!is_null($default))
            {
                $data = array($content_type_name, $default, $this-
>content_type_id);
                $data_types = array('text', 'integer', 'integer');
            }
            else
            {
                $data = array($content_type_name, $this-
>content_type_id);
                $data_types = array('text', 'integer');
            }

            DB::executePreparedQuery($sql, $data, $data_types,
"manip");

            $this->update();
        }
        catch (Exception $e)
        {
            return false;
        }
        return true;
    }

    /*
     * Delete the content type
     * Returns true (in case of success) or false (otherwise)
     */
    public function delete()
    {
        $sql = "
            DELETE FROM contents_types
            WHERE content_type_id = ?;";
        $data = array($this->content_type_id);
        $data_types = array('integer');
        try
        {
            DB::executePreparedQuery($sql, $data, $data_types,
"manip");
        }
        catch (Exception $e)
```

```
        {
            return false;
        }
        return true;
    }
}
?>
```

A.7 class.content_relation.php

```
<?php
include_once(dirname(__FILE__)."/class.instance.php");

/**
 *  This class maps every entry in the "contents_relations" table
 */
Class ContentRelation extends Instance
{
    static $instances = array();        // array of instances

    /**
     *  Implements get or create by ID
     */
    public static function findInstance($id, $parent_id, $data =
NULL)
    {
        return parent::findClassInstance(get_class(),
array("content_id"=>$id,"content_parent_id"=>$parent_id), $data);
    }

    /**
     *  Return the existance of a row (by its id)
     */
    public static function exists(array $ids)
    {
        $q = "SELECT * FROM contents_relations as cr WHERE 1";
        foreach($ids as $key => $value)
        {
            $q.= " AND cr.`".$key."` = ?";
            $data[] = $value;
        }
        $q.= " LIMIT 1;";
        $num = DB::executePreparedQueryAndGetNumRows($q, $data);
        return ($num>0) ? true : false;
    }

    /**
     *  Return an array of instances
     */
    public static function searchByContentId($id)
    {
        $q = "
            SELECT *
            FROM contents_relations as cr
            WHERE cr.content_id = ?
            ORDER BY cr.content_relation_parent_order ASC,
cr.content_parent_id ASC;";
```

```
        $data = DB::executePreparedQueryAndGetAssocArray($q,
array($id));
        $return_data = array();
        foreach ($data as $row)
        {
            $return_data[] =
ContentRelation::findInstance($row['content_id'],
$row['content_parent_id'], $row);
        }
        return $return_data;
    }

    /**
     *  Return an array of instances
     */
    public static function searchByContentParentId($id)
    {
        $q = "
            SELECT *
            FROM contents_relations as cr
            WHERE cr.content_parent_id = ?
            ORDER BY cr.content_relation_child_order ASC,
cr.content_id ASC;";
        $data = DB::executePreparedQueryAndGetAssocArray($q,
array($id));
        $return_data = array();
        foreach ($data as $row)
        {
            $return_data[] =
ContentRelation::findInstance($row['content_id'],
$row['content_parent_id'], $row);
        }
        return $return_data;
    }

    /**
     *  Instances a new object starting by its own id or, if
$data!=NULL, by the $data
     */
    public function __construct(array $ids,$data=NULL)
    {
        $classname = get_class($this);

        // Add new instance to static vector
        $classname::$instances[] = $this;

        if(is_null($data))
        {
            $q = "SELECT * FROM contents_relations as cr WHERE 1";
            foreach($ids as $key => $value)
            {
                $q.= " AND cr.`".$key."` = ?";
                $data[] = $value;
            }
            $q.= " LIMIT 1;";
            $data = DB::executePreparedQueryAndGetAssocArray($q,
$data);
            if (count($data))
                $data = $data[0];
```

```
            else
                throw new Exception("Impossibile trovare
content_relation con id <". join(", ", $data) .">.");
        }

        if (count($data))
        {
            foreach($data as $key => $item)
                $this->$key = $item;
        }
    }

    public function __get($name)
    {
        switch($name)
        {
            case "content":
                return $this->content = Content::findInstance($this-
>content_id);
                break;
            case "content_parent":
                return $this->content_parent =
Content::findInstance($this->content_parent_id);
                break;
        }
        trigger_error("Undefined property '$name' in " .
get_class($this));
    }

    public static function createRelation($content_id, $parent_id,
$parent_order = 0, $child_order = 0)
    {
        $q = "
            INSERT INTO contents_relations
            (content_id, content_parent_id,
content_relation_parent_order, content_relation_child_order)
            VALUES (?, ?, ?, ?)";
        $data = array($content_id, $parent_id, $parent_order,
$child_order);
        $data_types = array("integer", "integer", "integer",
"integer");
        DB::executePreparedQuery($q, $data, $data_types, "manip");

        return ContentRelation::findInstance($content_id,
$parent_id);
    }
}
?>
```

A.8 class.content_enum.php

```
<?php
include_once(dirname(__FILE__)."/class.instance.php");

/**
 *  This class maps every entry in the "contents_enums" table
 */
Class ContentEnum extends Instance
```

```
{
    static $instances = array();        // array of instances

    /**
     *  Implements get or create by ID
     */
    public static function findInstance($id, $data = NULL)
    {
        return parent::findClassInstance(get_class(),
array("content_enum_id"=>$id), $data);
    }

    /**
     *  Return the existance of a row (by its id)
     */
    public static function exists(array $ids)
    {
        $q = "SELECT * FROM contents_enums as ce WHERE 1";
        $data = array();
        foreach($ids as $key => $value)
        {
            $q.= " AND ce.`".$key."` = ?";
            $data[] = $value;
        }
        $q.= " LIMIT 1;";
        $num = DB::executePreparedQueryAndGetNumRows($q, $data);
        return ($num>0) ? true : false;
    }

    /**
     *  Return an array of instances
     */
    public static function searchByContentAttributeId($id)
    {
        $q = "
            SELECT *
            FROM contents_enums as ce
            WHERE ce.content_attribute_id = ?
            ORDER BY content_enum_order ASC;";
        $data = DB::executePreparedQueryAndGetAssocArray($q,
array($id));
        $return_data = array();
        foreach ($data as $row)
        {
            $return_data[] =
ContentEnum::findInstance($row['content_enum_id'], $row);
        }
        return $return_data;
    }

    /**
     *  Instances a new object starting by its own id or, if
$data!=NULL, by the $data
     */
    public function __construct($id,$data=NULL)
    {
        $classname = get_class($this);

        // Add new instance to static vector
```

```
        $classname::$instances[] = $this;

        if(is_null($data))
        {
            $q = "SELECT * FROM contents_enums as ce WHERE
ce.content_enum_id = ? LIMIT 1;";
            $data =
DB::executePreparedQueryAndGetAssocArray($q,array($id));
            if (count($data))
                $data = $data[0];
            else
                throw new Exception("Impossibile trovare
content_enum con id $id.");
        }

        if (count($data))
        {
            foreach($data as $key => $item)
                $this->$key = $item;
        }
    }
}
?>
```

Appendice B Progettazione della base di dati con MySQL Workbench

La progettazione della base di dati è stata eseguita utilizzando lo strumento offerto da MySQL denominato MySQL Workbench. Questo strumento permette di:

- stabilire una connessione con un particolare database;
- disegnare uno schema E-R;
- stabilire una connessione con il demone mysqld e amministrare il server MySQL.

La seguente figura mostra la schermata di avvio dell'applicazione. I tre riquadri mostrano le tre funzioni sopra citate.

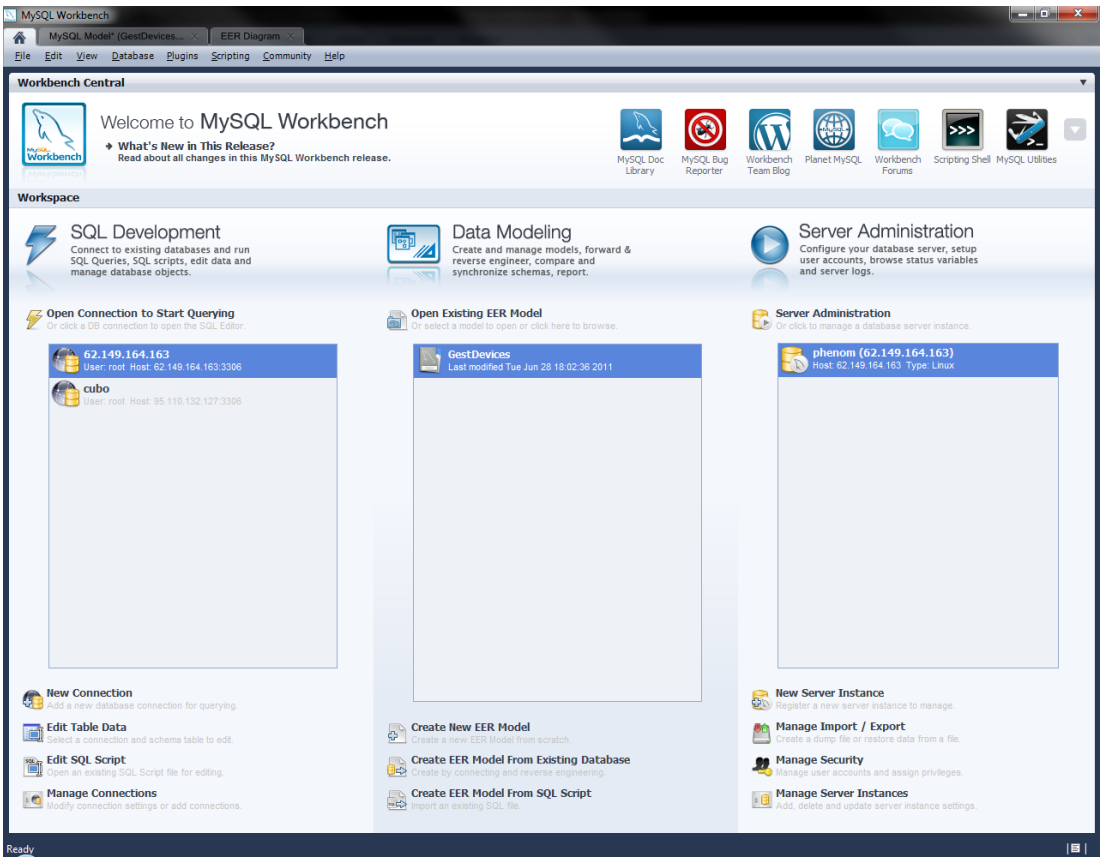


Figura 44 - Schermata di avvio di MySQL Workbench

B.1 Instanziare un server

La prima cosa da fare è instanziare un server attraverso la connessione al DB. In fase di sviluppo MySQL viene eseguito in locale quindi l'host di connessione è localhost sulla porta 3306.

Dopo aver creato l'istanza è possibile cambiare i parametri di configurazione come ad esempio il motore predefinito di memorizzazione. Quindi nel caso della base di dati che si vuole progettare viene impostato InnoDB.

B.2 Creare una connessione verso il DB

Tramite questo comando è possibile creare una nuova connessione al DB impostando un default schema, se già è stato definito. I parametri della connessione richiedono un host (localhost in fase di sviluppo) e delle credenziali di accesso. Ora è possibile:

- creare tabelle;
- inserire e visualizzare record;
- verificare la bontà delle query necessarie in fase di sviluppo software.

B.3 Disegnare uno schema E-R

Lo strumento di design di uno schema E-R permette di modellare una base di dati ex-novo oppure di utilizzare una base di dati esistente per creare un nuovo modello. Questo strumento è molto utile per verificare i legami fra le tabelle ed impostare quindi i vincoli di integrità referenziali. Inoltre risulta utile per avere una visione di insieme dello schema di una base di dati.

Appendice C La libreria MDB2

L'installazione della libreria può essere ottenuta da riga di comando lanciando il semplice comando:

```
pear install mdb2
```

L'aggiunta del driver mysql:

```
pear install mdb2#mysql
```

MDB2 si basa sull'utilizzo di DSN (Data Source Name) tipico di Java. La connessione al DB è ottenuta attraverso l'esecuzione di una funzione della classe MDB2:

```
<?php
    $mdb2 =& MDB2::singleton($dsn);
?>
```

dove $dsn è ad esempio:

```
<?php
    $dsn = "mysql://username:password@hostspec/database";
?>
```

La connessione è ottenuta attraverso la chiamata singleton e quindi permette di avere una MDB2_Driver_Common instance senza dover utilizzare un oggetto come globale.

Similmente a come avviene per le chiamate all'estensione MySQL di PHP, MDB2 offre dei metodi dell'oggetto MDB2 istanziato dalla singleton. I metodi utilizzati sono:

- $res =& $mdb2->query(): per processare una query;
- $res->fetchRow() o $res->fetchAll(): per processare l'oggetto result.

Inoltre è possibile utilizzare le transazione e sopratutto i metodi prepare e le execute. Questi metodi permettono di eseguire query frequentemente utilizzate, ma con diversi valori. Questi due metodi utilizzano il concetto di types per la loro esecuzione. I types sono un concetto molto simile a quello presente nei JavaBeans di Java e si rendono necessari per avere un'implementazione astratta dal DB e poter

fare inserimenti, cancellazioni e update nonché selezioni o altro. I types permettono la conversione dei tipi verso tutti i driver DB disponibili per questa libreria.

Una chiamata ad un metodo prepare è fatta in questo modo:

```
<?php
    $types = array('integer', 'text', 'text');
    $sth = $mdb2->prepare('INSERT INTO numbers VALUES (?, ?, ?)',
$types, MDB2_PREPARE_MANIP);

    $data = array(1, 'one', 'en');
    $affectedRows = $sth->execute($data);
?>
```

Un'altra funzionalità offerta dai metodi prepare() and execute() è il quoting dei valori. Questo viene fatto in automatico dal metodo stesso, così da evitare allo sviluppatore di doversi ricordare di applicare la funzione add_slashes() sulle stringhe che inserisce nelle sue query.

Appendice D Manuale dello sviluppatore per UniPOS

D.1 Core di sistema

D.1.1 File di configurazione

Il file di configurazione del sistema è ubicato in inc/inc.config.php e permette di impostare i parametri fondamentali:

```
// Generic section

define("IS_PRODUCTION", FALSE);
```

Impostando a TRUE il valore di IS_PRODUCTION vengono disabilitati tutti i messaggi di errore. Si consiglia di applicare modifiche al sistema, impostandone il valore a FALSE. Utilizzare invece TRUE quando il sistema è in produzione.

```
// Smarty section
define("SMARTY_DEBUGGING", !IS_PRODUCTION && FALSE);
define("SMARTY_FORCE_COMPILE", FALSE);
```

Le impostazioni di Smarty, utile strumento MVC, permettono di mostrare la console di debug delle variabili e forzare la ricompilazione dei template.

```
// Database section
define("DB_HOST", "localhost");
define("DB_USER", "gestdevice_it");
define("DB_PASS", "petsasabdo16");
define("DB_DB", "nome_database");
```

I parametri per la connessione al database vengono specificati in questa sezione.

```
// Session section
define("SESSION_LIFETIME", 0);
define("SESSION_DOMAIN", ".gestdevice.pisadinotte.it");
```

I parametri del cookie di session vengono impostati qui.

```
// Directories
define("LOGS_DIRECTORY", realpath(dirname(__FILE__)."/../logs/"));
define("JOBS_DIRECTORY", realpath(dirname(__FILE__)."/../jobs/"));
```

La cartella dove sono ubicati i file di log dei dispositivi e la cartella dove si trovano i file delle attività programmate.

```
// PHP executable
define("PHP_EXECUTABLE", "/usr/bin/php-cgi");
```

Indicare il percorso dell'eseguibile php. Si consiglia un percorso assoluto. Questo è necessario per la validazione della sintassi php nella creazione di una attività programmata.

D.1.2 Variabili persistenti

Le informazioni relative ai log dei dispositivi devono sopravvivere all'esecuzione di un'interprete, per permettere, ad esempio, di conteggiare quanti eventi di un certo tipo si sono verificati su un certo terminale.

Utilizzando le variabili persistenti è possibile memorizzare qualunque tipo di variabile disponibile in php: da semplici stringhe ad array molto complessi.

Le variabili persistenti sono disponibili in qualsiasi funzione personalizzata, semplicemente invocando i metodi statici riportati di seguito.

Ogni variabile viene riconosciuta tramite un identificativo che costituisce il primo parametro delle funzioni che seguono.

D.1.2.1 getProperty()

Questa funzione permette di reperire, in un punto qualsiasi dello script, il valore della variabile persistente specificata. La sintassi di questa funzione è la seguente:

```
$valore = GestDevice::getProperty($nome_variabile);
```

D.1.2.2 setProperty()

Questa funzione permette di salvare il valore di una variabile all'interno della variabile persistente specificata. La sintassi di questa funzione è la seguente:

```
GestDevice::setProperty($nome_variabile, $valore);
```

D.2 Utenti

GestDevice permette la gestione degli account utente e dei privilegi ad essi associati. Gli utenti vengono creati dagli amministratori di sistema compilando un form di richiesta composto dai campi per l'identificazione e un campo per la gestione dei privilegi.

Le tipologie di utenti previste sono, nell'ordine di importanza e livello di accesso alle funzionalità:

- SuperAdmin
- Sviluppatore
- Amministratore
- Amministratore di Facoltà
- Operatore

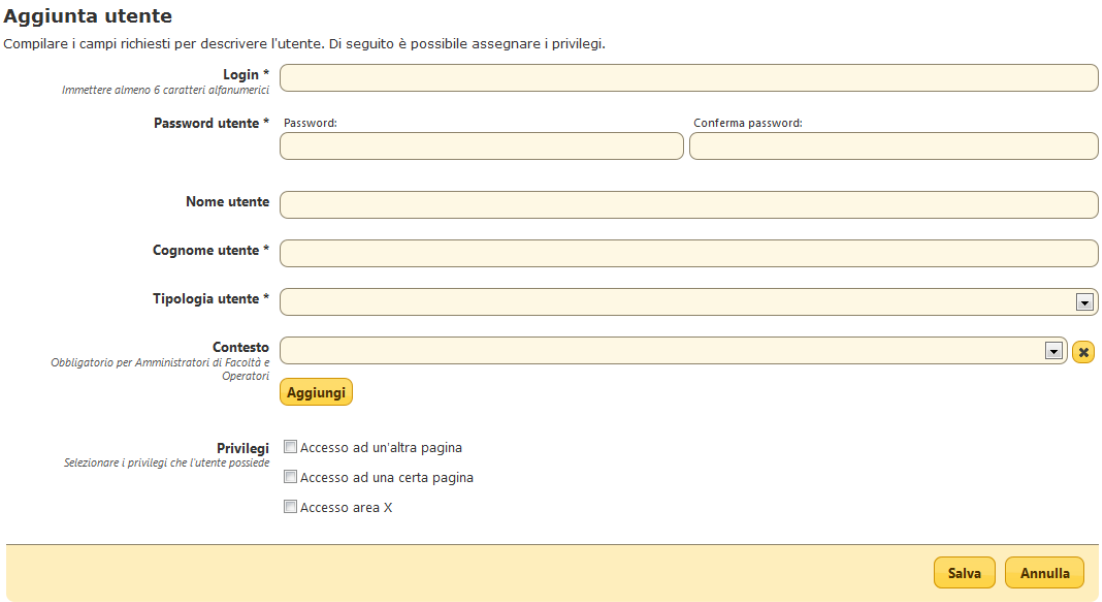
D.2.1 Creazione di un utente

La creazione di un utente è disponibile alla pagina:

Utenti > Aggiungi utente.

La pagina mostra un form di richiesta con i seguenti campi:

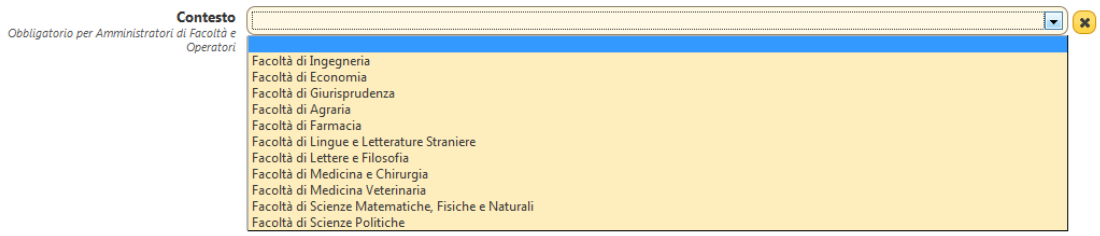
- Login (obbligatorio);
- Password e conferma password (obbligatorio);
- Nome utente;
- Cognome utente (obbligatorio);
- Tipologia utente (obbligatorio);
- Contesto (obbligatorio se la tipologia utente selezionata è Amministratore di Facoltà oppure Operatore).



Il campo “tipologia utente” viene compilato con la scelta fra le possibili tipologie che dipendono dal tipo account dell'utente che sta eseguendo l'operazione.

Account che esegue l’operazione	Tipologia assegnabile
SuperAdmin	Amministratore Amministratore di Facoltà Sviluppatore Operatore
Sviluppatore	Amministratore Amministratore di Facoltà Operatore
Amministratore	Amministratore di Facoltà Operatore
Amministratore di Facoltà	Operatore
Operatore	-

Il campo “contesto” mostra un elenco dei figli di primo livello della struttura organizzativa (vedi l'appendice D.3.4) e permette di assegnare una o più facoltà: l’accesso dell’utente che si sta per creare avrà quindi funzionalità limitare alle sole facoltà che gli sono state assegnate.



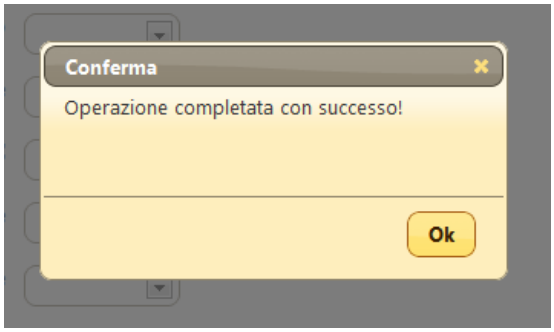
Un utente di un certo livello è in grado di creare nuovi utenti solo delle tipologie inferiori alla propria, secondo i livelli indicati a inizio capitolo.

I privilegi che l’utente corrente può assegnare sono esclusivamente quelli che possiede, se l’utente corrente non è almeno di livello Amministratore. Per questo motivo la modifica di un utente di livello inferiore, attualmente in possesso di privilegi che non rientrano in quelli dell’utente corrente, comporterebbe la perdita di tali privilegi ed è pertanto interdetta. Allo stesso modo, non è possibile

modificare un operatore che appartiene ad un contesto più ampio di quello amministrato dall'utente corrente (fanno eccezione gli Amministratori, gli Sviluppatori e l'utente Super Amministratore).

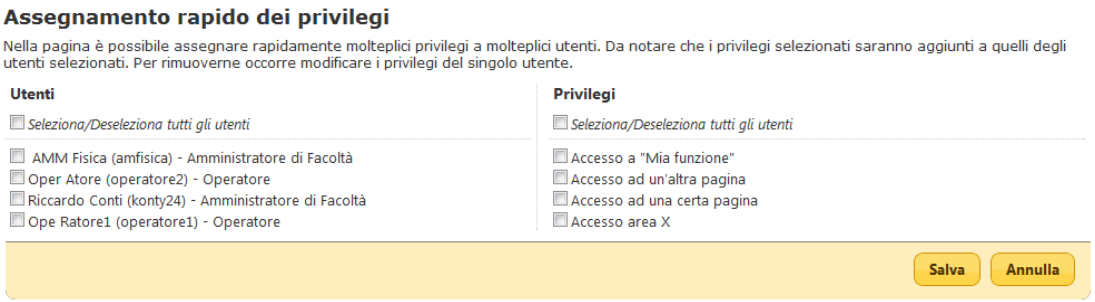
Una volta compilati i campi l'utente deve premere "Salva". Altrimenti se decide di annullare l'operazione può premere sul tasto "Annulla".

Al completamento dell'operazione compare a schermo questo avviso:



D.2.2 Assegnamento dei privilegi

L'assegnamento dei privilegi può avvenire sia per singolo utente che per utenti multipli: al fine di renderne più rapido l'assegnamento a molteplici utenti, è stata predisposta una funzione speciale che consente di selezionare quali privilegi assegnare a quali utenti.



È disponibile una pratica funzione di selezione rapida di tutti gli utenti o di tutti i privilegi.

Da notare che questa operazione permette di aggiungere permessi, non di rimuoverli. Per rimuovere un privilegio ad un utente è necessario utilizzare la scheda di modifica utente e quindi deselezionare i privilegi che si desidera revocare (come illustrato nell'appendice D.2.1).

D.2.3 Limitazione di accesso alle funzioni personalizzate

La realizzazione di nuove funzionalità spesso incontra la necessità di restringerne l'accesso soltanto a pochi utenti. Per implementare con successo tale politica è sufficiente creare (vedi l'appendice D.5.5) ed assegnare un privilegio personalizzato ad ogni utente a cui si desidera concedere l'accesso. Nell'appendice D.5.3 viene illustrato come limitare l'accesso a funzioni (comandi) di menu e nell'appendice D.5.4 viene illustrata la procedura di limitazione di accesso alle funzioni non di menu.

D.3 Dispositivi

GestDevice è una piattaforma che permette la gestione di dispositivi all'interno di un'organizzazione. Ogni dispositivo è caratterizzato da un tipo che raccoglie l'insieme degli attributi che definiscono le peculiarità e le funzionalità utili per la descrizione completa del dispositivo stesso.

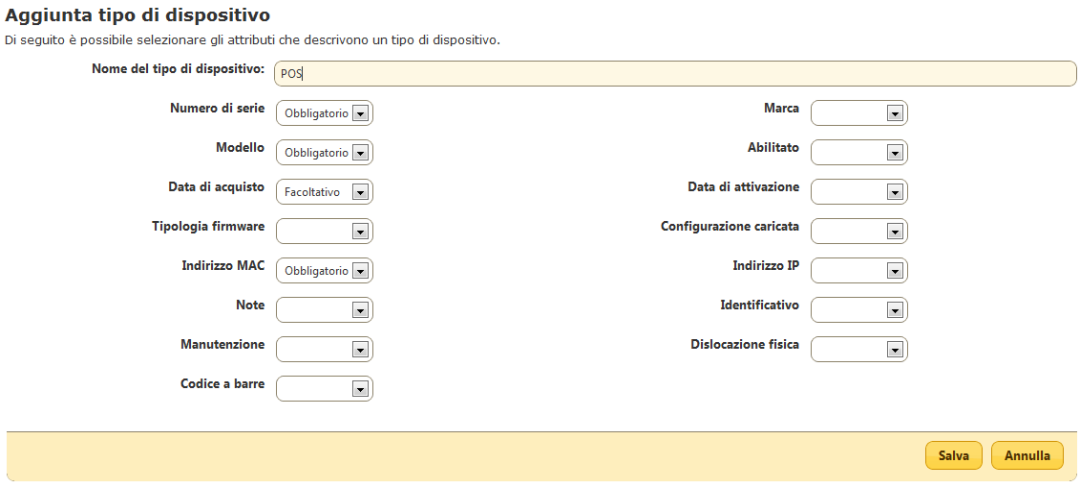
D.3.1 Creazione di un tipo di dispositivo

L'utente, che utilizza l'interfaccia per la gestione dei dispositivi, può creare un tipo di dispositivo accedendo alla sezione:

Dispositivi > Aggiungi tipo di dispositivo

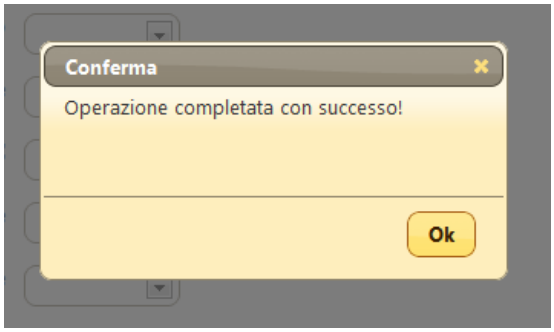
Da qui l'utente deve inserire il nome del tipo di dispositivo e selezionare gli attributi che lo descrivono. Per ognuno di questi può selezionare:

- **Campo vuoto**: l'attributo non viene aggiunto al tipo e alla descrizione del dispositivo;
- **Obbligatorio**: l'attributo risulta obbligatorio e dovrà essere compilato per la descrizione del dispositivo;
- **Facoltativo**: l'attributo è facoltativo e può non essere compilato per la descrizione del dispositivo.



Un volta completata l'operazione l'utente deve premere su "Salva". Altrimenti, se decide di annullare l'operazione, può premere su "Annulla".

Una volta completata l'operazione compare a schermo questo avviso:



D.3.2 Creazione di un attributo

L'inserimento di un tipo di dispositivo prevede la scelta degli attributi per la compilazione della scheda di un dispositivo. L'utente visualizza una lista di attributi di default (quelli per la descrizione di un dispositivo POS), ma può richiedere all'amministratore di sistema l'inserimento di attributi custom.

D.3.2.1 Creazione di un gestore di un tipo di attributo

L'inserimento di un attributo personalizzato viene eseguito dall'amministratore di sistema. L'aggiunta di un attributo prevede due operazioni:

- Inserimento in database di un record;
- Scrittura di un plugin in PHP per la gestione dell'attributo.

L'inserimento in database si configura come segue:

content_attribute_id	content_attribute_name	content_attribute_default	content_attribute_type	content_attribute_identificator	content_attribute_options	content_attribute_tags
61	Marca	1	free_text	brand	text_type=single_row	device

Nell'esempio proposto si notano i campi:

- content_attribute_id: valore assegnato automaticamente dal sistema, identificante l'attributo;
- content_attribute_name: l'etichetta che descrive l'attributo; tale etichetta viene mostrata nel form di inserimento/modifica di un dispositivo;
- content_attribute_default: solitamente questo valore deve essere lasciato a 0, in quanto l'attributo che si sta creando è personalizzato;
- content_attribute_type: una stringa composta da [a-z0-9_] che identifica il tipo di attributo; il valore "free_text" indica che l'attributo è un campo di testo libero;
- content_attribute_identificator: si tratta di una stringa composta da [a-z0-9_] che permette di identificare l'attributo dentro l'oggetto (es. $object->brand);
- content_attribute_options: una stringa in formato "url_encoded" che permette di specificare parametri sul metodo di salvataggio e validazione del campo (es. text_type=single_row indica che il controllo di input dell'attributo dovrà essere mostrato come singola linea, piuttosto che come area di testo con righe multiple;
- content_attribute_tags: data la possibilità di rappresentare più tipologie di oggetti, potrebbe essere necessario mostrare, durante la creazione di un tipo di oggetto, solo gli attributi che rispondono a determinati tag; i valori inseribili sono a piacimento e occorre scriverli nel formato CSV.

Creato l'attributo, qualora non si sia scelto un tipo di attributo già gestito, occorre crearne uno che sia in grado di interpretare, validare e salvare le informazioni che lo descrivono. A titolo di esempio si riporta il codice del gestore del tipo "free_text":

```
<?php
include_once(dirname(__FILE__)."/../class.attribute_type_manager.php
");

Class FreeTextManager extends AttributeTypeManager
{
    public static function canHandle(ContentAttribute $attribute)
```

```
    {
        if($attribute->content_attribute_type=="free_text")
            return true;
        return false;
    }

    public function parseValue(&$destination_property,
ContentAttributeValue $value)
    {
        $destination_property = $value->text;
    }

    public function objectConstructor(ContentAttributeValue $value)
    {
        $value->text = $value->content_attribute_value_value;
    }

    public function validateValue($value, $mandatory = false)
    {
        if($mandatory)
            return ($value === "") ? "missing" : true;
        else
            return true;
    }

    public function adminTemplate($values = "")
    {
        global $default;
        $text_type = isset($this->content_attribute-
>options['text_type']) ? $this->content_attribute-
>options['text_type'] : "single_row";
        switch($text_type)
        {
            case 'multiple_row':
                $o = "<table id=\"attribute_".$this-
>content_attribute->content_attribute_identificator."\"
cellspacing=\"0\" cellpadding=\"0\"
class=\"attribute\"><tbody><tr><td><textarea class=\"text\"
name=\"".$this->content_attribute-
>content_attribute_identificator."\" type=\"text\">" .
htmlentities($values, ENT_QUOTES, "UTF-8") . "</textarea><div
id=\"error_".$this->content_attribute-
>content_attribute_identificator."\"
class=\"error\"></div></td></tr></tbody></table>";
                break;
            case 'single_row':
            default:
                $o = "<table id=\"attribute_".$this-
>content_attribute->content_attribute_identificator."\"
cellspacing=\"0\" cellpadding=\"0\"
class=\"attribute\"><tbody><tr><td><input class=\"text\"
name=\"".$this->content_attribute-
>content_attribute_identificator."\" type=\"text\" value=\"" .
htmlentities($values, ENT_QUOTES, "UTF-8") . "\" /><div
id=\"error_".$this->content_attribute-
>content_attribute_identificator."\"
class=\"error\"></div></td></tr></tbody></table>";
                break;
        }
```

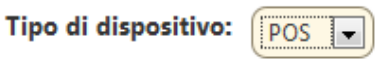
```
        return $o;
    }

    public function insertValue($value)
    {
        return $value;
    }

    public static function allowedTypes(&$arr)
    {
        $arr['free_text'][] = array(
            "options" => "text_type=single_row",
            "label" => "Testo su linea singola"
        );
        $arr['free_text'][] = array(
            "options" => "text_type=multiple_row",
            "label" => "Testo su linea multipla"
        );
    }

}
?>
```

Analizziamo le funzioni membro proposte:

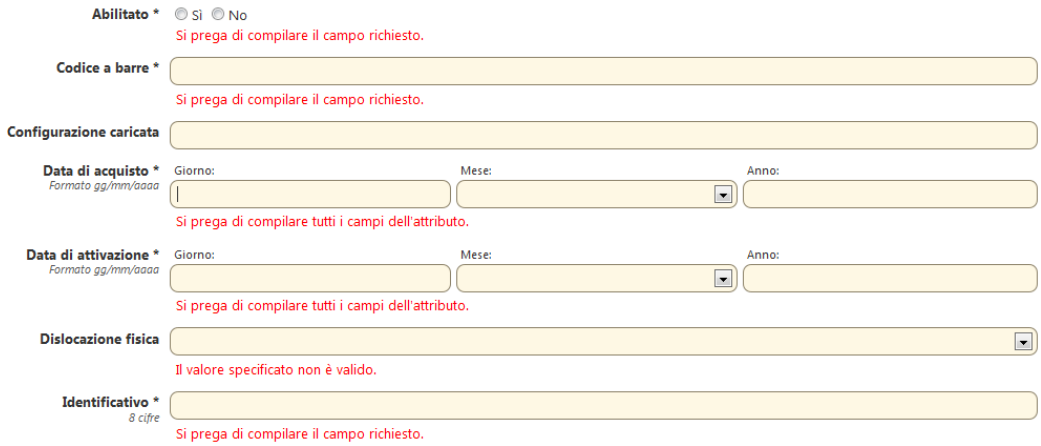
- *canhandle(ContentAttribute $attribute)*: ritorna al sistema un booleano indicante la sua abilità a gestire il tipo di attributo in questione;
- *parseValue(&$destination_property, ContentAttributeValue $value)*: si occupa di popolare l'oggetto separando le informazioni essenziali a descrivere l'oggetto;
- *objectConstructor(ContentAttributeValue $value)*: si occupa di popolare gli attributi membro l'oggetto;
- *validateValue($value, $mandatory = false)*: permette di verificare che i dati passati dall'utente nella compilazione del form siano corretti; in caso contrario restituisce un messaggio di errore;
- *adminTemplate($values = "")*: si occupa di generare il modello grafico da mostrare per mostrare all'utente il campo/i campi da compilare; l'argomento permette di compilare i campi come valori già presenti in database;
- *insertValue($value)*: si occupa di ritornare una stringa da inserire in database per descrivere ciò che l'utente ha inserito;
- *allowedTypes(&$arr)*: in base alle opzioni disponibili gestite dal plugin si possono specificare tipologie di valori di campi consentiti, fornendo anche una descrizione.

D.3.3 Creazione di un dispositivo

La schermata di aggiunta di un dispositivo permette all'utente di scegliere di quale tipo di dispositivo si tratta, attraverso il controllo mostrato di seguito:



Una volta scelto il tipo di dispositivo, vengono mostrati gli attributi che lo descrivono ed è quindi possibile inserire le caratteristiche peculiari del dispositivo che si sta inserendo. Ogni campo è sottoposto a validazione da parte del gestore di attributo. Se il formato dei dati inseriti non risultasse corretto, vengono mostrati degli avvisi subito sotto i campi non conformi.



Nell'immagine appena sopra si nota che, ad esempio, il campo "Configurazione caricata" non sia obbligatorio e risulta quindi privo di errori, anche se vuoto.

Cliccando sul pulsante "Salva" si conclude la procedura di creazione del dispositivo e si fa ritorno alla lista dei dispositivi.

D.3.4 Gestione della struttura organizzativa

La gestione della struttura organizzativa permette di specificare i livelli in cui è strutturata una organizzazione, ed è raggiungibile attraverso il percorso:

Dispositivi > Gestione struttura organizzativa

La struttura è estendibile ad infiniti livelli di dettaglio, utilizzando gli appositi comandi di aggiunta di un livello.

Il nome di ogni livello è modificabile premendo sul pulsante di modifica che si trova sulla barra di ogni livello. È sempre possibile rimuovere un livello, tranne nel caso in cui sotto di esso si trovino altri livelli: la motivazione di questo risiede nella necessità di evitare l'involontaria cancellazione di informazioni necessarie alla localizzazione dei dispositivi.

D.4 Attività programmate

GestDevice permette la gestione di attività programmate da mandare in esecuzione ad intervalli definiti dall'utente. Ogni minuto il sistema verifica:

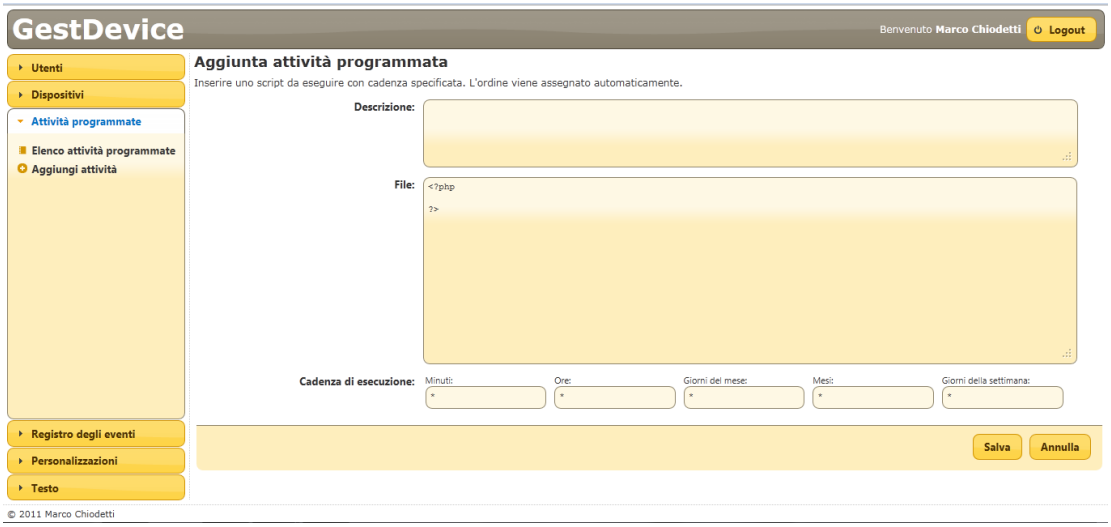
1. La presenza di file di log nella cartella specificata nell'istruzione *define(LOGS_DIRECTORY)* nel file *inc/inc.config.php* ed esegue gli interpreti opportuni (vedi l'appendice D.6);
2. Schedula le attività programmate, secondo le regole di esecuzione.

D.4.1 Creazione di un'attività programmata

L'utente per poter aggiungere una nuova attività programmata deve accedere alla pagina:

Attività programmate > Aggiungi attività

Viene mostrato il form di inserimento:



I campi da compilare sono:

- Il campo descrizione;
- Script da eseguire;
- Cadenza di esecuzione.

Il campo “cadenza di esecuzione” deve essere compilato seguendo la sintassi del comando cron di UNIX.

Il campo “file” contiene la logica che deve essere eseguita per un task e può fare ricorso a tutte le funzioni o classi definite nel sistema. In particolare vengono utilizzati i metodi *getProperty()* e *setProperty()* della classe GestDevice (vedi l'appendice D.1.2).

D.4.1.1 Sintassi cron

Il comando cron prevede questa sintassi per la gestione della cadenza di esecuzione di uno script:

```
*  *  *  *  * <comando>
```

L’asterisco è un operatore che definisce la cadenza di esecuzione dello script personalizzandola attraverso 5 valori:

- minuti;
- ore;
- giorni del mese,
- mese;
- giorni della settimana.

Per ogni campo è possibile specificare valori multipli attraverso gli operatori:

- operatore virgola: specifica una lista di valori;
- operatore trattino: specifica un intervallo di valori;
- operatore asterisco: specifica tutti i valori di un campo. Ad esempio un asterisco nel campo ora è equivalente ad "ogni ora".

Un esempio di esecuzione è il seguente:

```
0,20,40 8-17 * * 1-5
```

L'esecuzione avviene ogni venti minuti dalle 8 alle 17 di ogni giorno feriale, in tutti i mesi e in tutti i giorni del mese.

D.5 Personalizzazione

D.5.1 Gestione dei comandi del menu

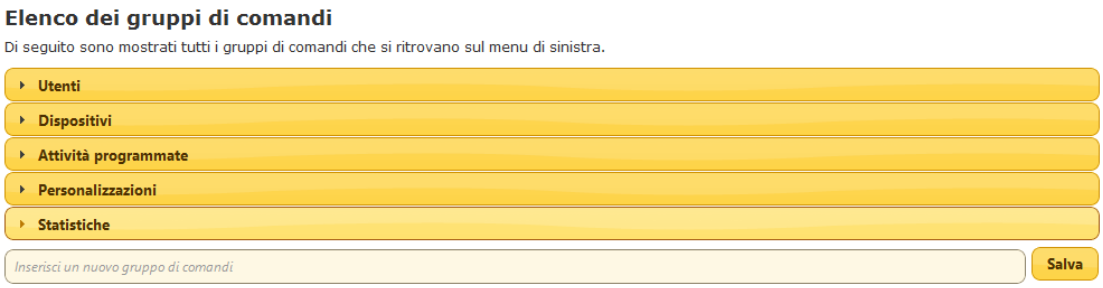
Al fine di poter personalizzare il comportamento del sistema sulla base di esigenze specifiche, è stato previsto un meccanismo che consente di gestire il menu laterale sinistro, generando nuovi contenitori e nuove pagine.

Nella figura precedente è stato creato un gruppo predefinito denominato “Mio gruppo”, con un comando personalizzato denominato “Rapporto dispositivi”.

D.5.2 Creazione di un gruppo di comandi

L'aggiunta di nuove funzionalità al sistema necessita il raggruppamento delle stesse per aree tematiche: comandi inerenti lo stesso ambito possono essere visualizzati entro il medesimo riquadro di comando. Per creare un gruppo di comandi è necessario seguire:

Personalizzazioni > Gestione comandi di menu



Compilando il campo in calce alla pagina è possibile specificare un nome per il nuovo gruppo di comandi. Cliccando sul pulsante di salvataggio, il nuovo comando sarà immediatamente visibile nel menu di sinistra.

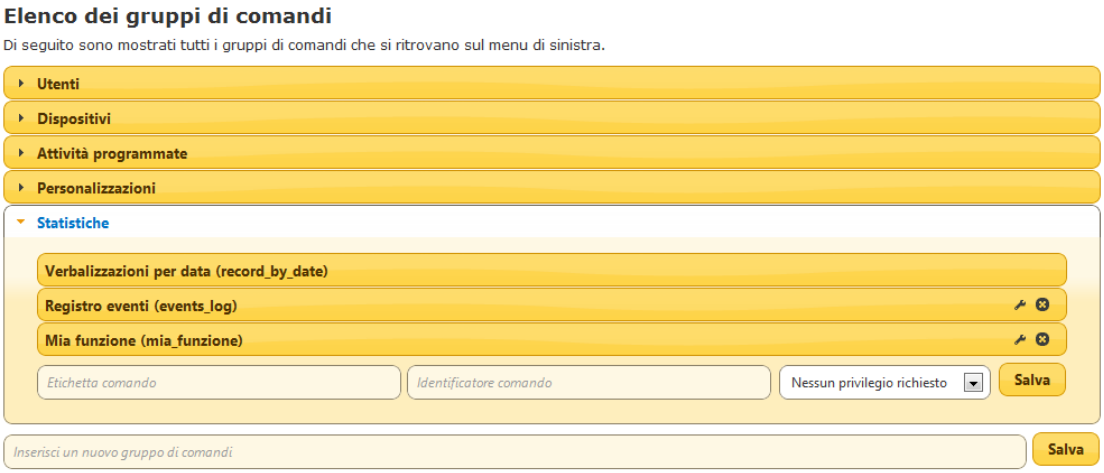
Da notare che, accedendo come Amministratore di Facoltà oppure Operatore, il menu di sinistra verrà sfoltito delle categorie che contengono tutti comandi cui l'utente non può accedere.

D.5.3 Creazione di un comando

La creazione di una nuova funzionalità inizia da qui: innanzitutto il comando deve essere raggiungibile dall'utente e quindi occorre posizionarlo entro un gruppo di comandi. Occorre accedere in:

Personalizzazioni > Gestione comandi di menu

Cliccando su un gruppo, questo si espande, mostrando i comandi presenti.



Sono presenti sia comandi predefiniti, facilmente identificabili perché sprovvisti dei comandi di modifica e rimozione, sia comandi personalizzati. Per questi ultimi occorre specificare, sia in creazione che in modifica:

- etichetta del comando;
- identificatore del comando: stringa contenente i caratteri [a-z0-9_] che rappresenta il comando nel codice sorgente;
- privilegio richiesto: specificando un privilegio, solo gli utenti che lo possiedono sono in grado di accedervi.

Più in dettaglio, qualora si desideri che un utente non sia in grado di visualizzare/accedere nel menu una certa funzionalità occorre specificare, nel campo "privilegio richiesto", il privilegio che l'utente deve possedere. Se il campo

"privilegio richiesto" non viene compilato, il nuovo comando sarà visualizzabile ed accessibile da qualsiasi tipologia di utente.

D.5.3.1 Scrittura del codice che implementa il comando

La logica che implementa il comando appena creato è necessario realizzarla: l'operazione richiesta è quella di creare un file denominato *cmd.identificatore_comando.php* nella cartella */custom_commands*.

Non è necessario specificare alcun codice che limiti l'accesso, in quanto il sistema provvede in automatico a verificare che l'utente sia provvisto del privilegio richiesto (specificato nel campo "privilegio richiesto" durante la creazione del comando).

Il codice scritto in questo file viene considerato con un case dello switch presente in *ajax.php*: sono pertanto utilizzabili tutte le variabili disponibili in questo contesto.

In particolare occorre tenere presente la variabile essenziale:

```
<?php
    $response = array(
        'cmd' => "",
        'user' => array(
            'logged' => false,
            'data' => array()
        ),
        'html' => ""
    );
?>
```

La chiave che ci interessa maggiormente è "html", in cui si specifica il codice html da innestare nella pagina (contenitore #main). L'innesto avviene in modo automatico.

D.5.3.1.1 Limitare l'acceso alle funzionalità

Di seguito viene mostrata una tabella che elenca l'ordinamento degli utenti, in ordine di livello di accesso alle funzionalità:

Identificativo utente	Tipologia utente	Ordinamento
7	Super Amministratore	0
6	Sviluppatore	1
4	Amministratore	2
8	Amministratore di Facoltà	3

5	Operatore	4

Per impedire l'accesso alla funzionalità che si sta realizzando, occorre inserire, in testa allo script, un breve controllo, esemplificato nel seguente snippet di codice:

```
<?php
    // Set the minimum access level to the Admin (order = 2)
    if(Utility::lockDown(2, $response))
        return;

    // My script...
?>
```

Il livello minimo richiesto per eseguire questa operazione è Amministratore. Ciò significa che solo gli utenti Amministratore, Sviluppatore e Super Amministratore possono eseguire lo script. Impostando il valore a 4, oppure non includendo questo comando, l'accesso non viene ristretto.

La variabile $response deve essere sempre passata per permettere al sistema di restituire il messaggio di errore, qualora l'accesso sia impedito.

D.5.3.1.2 Limitare l'accesso alle informazioni in base al contesto

Per limitare l'accesso solo alle informazioni dei dispositivi che stanno sotto un certo nodo della struttura organizzativa (es. Facoltà di Ingegneria), può essere utilizzata la seguente funzione:

```
<?php
    Class Utility
    {
        //...
        public static function canAccessDevice($device_id)
        {
            $device = Device::findInstance($device_id);
            if(is_null($device))
                return false;

            if(Utility::checkPrivilegeByUserTypeOrder(2))
                return true;

            if(!isset($device->parsed_attributes['location']))
                return false;

            return Utility::canAccessPlace(
                  $device->parsed_attributes['location']['id']
            );
        }
        //...
```

```
    }
?>
```

Specificando un identificativo (il campo ID) di dispositivo, la funzione verifica che l'utente corrente sia in grado di accedere alle sue informazioni. La funzione ritorna un booleano.

Allo stesso modo, possiamo verificare se un utente è in grado di accedere ad informazioni vincolate ad un contesto (es. una entità che deve essere accessibile solo se si trova nel contesto della Facoltà di Ingegneria). La funzione che segue permette di verificare questa condizione:

```
<?php
    Class Utility
    {
        //...
        public static function canAccessPlace($place_id)
        {
            global $user;

            if(!isset($user->allowed_places_ids))
            {
                $allowed_roots_ids = array_map(
                    "RootLocationsManager::getLocationID",
                    $user->parsed_attributes['context']
                );
                $user->allowed_places_ids = array();
                foreach($allowed_roots_ids as $id)
                {
                    $first_level_place = Content::findInstance($id);
                    $descendents = $first_level_place-
>descendent_ids;
                    $user->allowed_places_ids = array_merge(
                        $user->allowed_places_ids,
                        $descendents
                    );
                }
            }

            if(in_array($place_id, $user->allowed_places_ids))
                return true;
            return false;
        }
        //...
    }
?>
```

D.5.4 Creazione di un comando non di menu

Una volta creato un comando di menu, si può avere la necessità di realizzare comandi che non si trovano necessariamente nel menu: per esempio, un comando

di menu può mostrare un form da compilare e un comando può occuparsi di gestire la risposta del server dopo che l'utente ha confermato la sua compilazione. Questo comando, quindi, non deve essere inserito nel menu.

Creiamo allora un nuovo file di logica nella cartella */custom_commands* e denominiamolo in modo che sia descrittivo dell'operazione che gestisce: *cmd.nome_comando.php*.

In questo caso la limitazione di accesso deve essere forzata, impostando queste due righe di codice:

```
<?php
    if(Utility::lockByPrivilege($privilege_id, $response)
        return;

    // Place your code here
?>
```

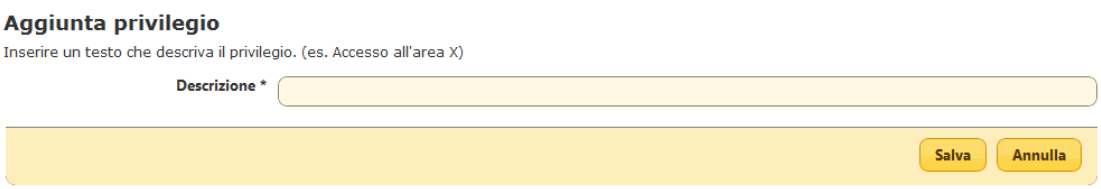
Ciò che la funzione richiede è il passaggio dell'identificativo del privilegio (ottenibile nella procedura descritta nell'appendice D.5.5) e la variabile globale che si occupa di gestire la risposta JSON.

Al di sotto di questo breve frammento di codice si posiziona la logica che implementa la funzione richiesta.

D.5.5 Creazione di un privilegio personalizzato

La limitazione di accesso ad una determinata funzione esige la creazione di un privilegio personalizzato:

Personalizzazioni > Aggiungi privilegio



L'interfaccia richiede all'utente una descrizione del privilegio che permette di identificare con facilità quale funzionalità vogliamo limitare.

Il sistema fornisce un identificativo univoco, da utilizzare nel codice illustrato nell'appendice D.5.4.

D.6 File di log ed interpreti

D.6.1 Impostare la cartella dei file di log

La cartella dove si trovano i file di log, che il sistema analizza ad intervalli di 1 minuto, deve essere specificata nel file di configurazione (vedi l'appendice D.1.1). Questa non è vincolata a trovarsi nella cartella predefinita del sistema.

D.6.2 Creazione di un interprete

Per aggiungere un interprete personalizzato è necessario scrivere un file php nella cartella class/interpreters, denominandolo class.nome_mio_interprete.php, a piacimento.

Di seguito viene mostrato un esempio di interprete:

```
<?php
include_once(dirname(__FILE__)."/../class.interpreter.php");

Class SimpleInterpreter extends Interpreter
{
    public static function canParse(&$string)
    {
        // Check if the given string can be parsed by this
interpreter
        // eg.
        // Prop1=val1
        // Prop2:Val2
        // Prop3 = val3
        // Prop4 : val4

        $re = "/\A\s*(\b([_\w]+)\s*[:=]\s*([- _\w]*?)\s*)+\Z/ms";

        if (preg_match($re, $string))
            return true;
        return false;
    }

    public static function parse(&$string)
    {
        // Parse the given string
        $re = "/^\s*([_\w]+)\s*[:=]\s*([- _\w]*?)\s*$/im";

        $matches = array();
        preg_match_all($re, $string, $matches, PREG_SET_ORDER);

        $attr = array();
```

```
        foreach($matches as $match)
        {
            $attr[strtolower($match[1])] = $match[2];
        }

        // Evaluates error status
        if (strtolower($attr['status'])==="error")
        {
            try
            {
                $x =
GestDevice::getProperty("errors_".$attr['serial_number']);
            }
            catch(Exception $e)
            {
                $x = 0;
            }

GestDevice::setProperty("errors_".$attr['serial_number'], ++$x);
            echo "Attenzione il dispositivo ".$attr['brand']."
".$attr['model']." ha generato ".$x." Errori.\n";

        }
    }
}
?>
```

L'esempio proposto è un interprete che è in grado di interpretare un generico file nel formato:

```
chiave=valore
```

Il metodo *canParse()* verifica se questo interprete è abilitato alla lettura del file passato come parametro. Essa ritorna un valore booleano che indica al sistema se chiamare il metodo parse() dello stesso interprete, oppure passare al prossimo.

Nel metodo *parse()* sono stati esplosi i campi ed inseriti in un array associativo: utilizzando una chiave è possibile accedere al valore presente nel file originale. In questo esempio è stata recuperata la variabile persistente (vedi l'appendice D.1.2) di un dispositivo con un certo numero seriale e viene incrementato di uno il valore della variabile che conteggia il numero di errori, qualora il file di log riporti un errore.

Infine memorizza nuovamente la variabile persistente, col medesimo identificatore, e mostra un messaggio di notifica. In questo esempio si nota chiaramente la potenzialità del meccanismo delle variabili persistenti.

Sitografia

- http:// php.net
- http://database.html.it/guide/lezione/2458/transazioni-e-lock/
- http://unipos.unipi.it/unipos.info/area_informativa.php
- http://adminschoice.com/crontab-quick-reference
- http://php.net/manual/en/function.php-check-syntax.php
- http://jqueryui.com/demos/accordion/
- http://jqueryui.com/demos/dialog/
- http://pear.php.net/package/MDB2
- http://docs.jquery.com/Main_Page
- http://meyerweb.com/eric/tools/css/reset/
- http://usabilita.bazzmann.com/introduzione/introduzione_01.php
- http://www.webmarketinguniversity.net/articolo/102/Il-layout-di-sito-web-ideale-per-usabilitc3a0-e-conversioni.htm
- http://www.smarty.net
- http://www.w3schools.com/browsers/browsers_stats.asp